BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 15/78

A2

(11) International Publication Number: **WO 00/22546**

(43) International Publication Date: 20 April 2000 (20.04.00)

(21) International Application Number: PCT/US99/24114

(22) International Filing Date: 13 October 1999 (13.10.99)

(30) Priority Data:
09/172,918 14 October 1998 (14.10.98) US

(71) Applicant: TRISCEND CORPORATION [US/US]; 301 N. Whisman Road, Mountain View, CA 94043 (US).

(72) Inventors: WINEGARDEN, Steven, Paul; 1063 Silver Tip Way, Sunnyvale, CA 94086 (US). REYNOLDS, Bart; 1668 East Boston Terrace, Seattle, WA 98112 (US). FOX, Brian; 865 Fremont Street, Santa Clara, CA 95050 (US). ALLEGRUCCI, Jean-Didier; 658 Cheshire Way, Sunnyvale, CA 94087 (US). KRISHNAMURTHY, Sridhar; 4321 Verdigris Circle, San Jose, CA 95134 (US). TAVANA, Danesh; 2524 Fairbrook Drive, Mountain View, CA 94040 (US). ZIKLIK, Arye; 1391 Lewiston Drive, Sunnyvale, CA 94087 (US). PAPALIOLIOS, Andreas; 1338 Torrance Avenue, Sunnyvale, CA 94089 (US). YANG, Stanley, S.; 1574 Honeysuckle Place, Los Altos, CA 94024 (US). LEE, Fung, Fung; 1213 Moulton Drive, Milpitas, CA 95035 (US).

(74) Agents: VINCENT, Lester, J. et al.; Blakely, Sokoloff, Taylor & Zafman, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025-1026 (US).

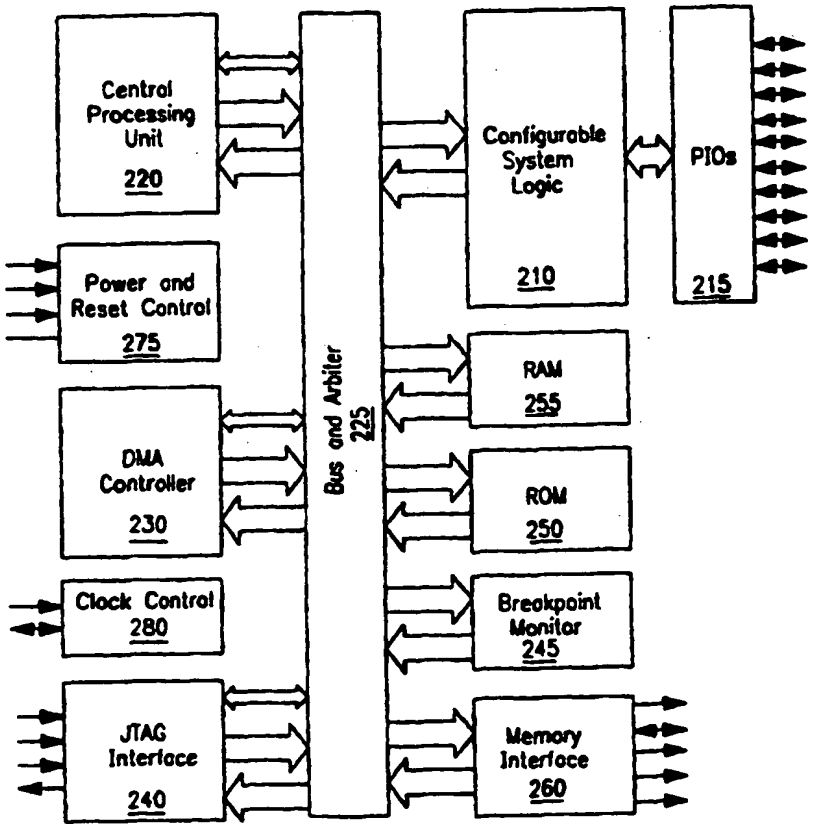
(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: CONFIGURABLE PROCESSOR SYSTEM UNIT

(57) Abstract

The configurable processor system includes a processor, an internal system bus, and a programmable logic all interconnected via the internal system bus, on a single integrated circuit.



Configurable Processor System Unit

Field of the Invention

The present invention relates to integrated circuits, and more specifically, to a configurable processor system unit.

Background of the Invention

Micro-controllers have been around over two decades dating back to the mid 1970's. A micro-controller describes a class of embedded systems. It is usually a single chip device containing a central processing unit, or CPU, custom logic, and an embedded program written in assembly or machine language. The CPU's program and the custom logic embedded within the micro-controller thus tailor its application. A typical calculator for example has a micro-controller specifically designed to interact with a LCD display, numeric keypad, and provide various arithmetic functions. From the inception of the micro-controller device, hundreds of different custom configurations have been designed to address specific applications. The overhead, in terms of time and cost, associated with customizing a micro-controller is significant. Therefore, custom micro-controllers make practical economic sense when large volumes, usually millions of units, are at stake.

An alternative to custom micro-controllers is an Application Specific IC, commonly known as ASIC. An ASIC offers user customization through personalizing the masking layers during the production of the integrated circuit device. Personalization usually comes in two forms: metal only mask, known as Gate-Arrays, and all mask personalization, called Standard Cells. The average ASIC user must overcome numerous obstacles and challenges before an actual product can be realized. Firstly, designing a CPU and associated development tools is a tremendous barrier for most customers. Secondly, the hardware and software expertise, development time, and non-recurring expenses associated with developing such an ASIC is usually much greater than the return. And

Figure 1 is a CPSU in a typical system environment.

Figure 2 is a block diagram of one embodiment of a CPSU architecture.

Figure 3 is a block diagram of one embodiment of bus segments.

Figure 4 is a timing diagram of one embodiment of an addressed write with no wait states.

Figure 5 is a timing diagram of one embodiment of an addressed write with one wait state.

Figure 6 is a timing diagram of one embodiment of an addressed read with no wait states.

Figure 7 is a timing diagram of one embodiment of an addressed read with two wait states.

Figure 8 is a timing diagram of one embodiment of a DMA acknowledge write without wait states present.

Figure 9 is a timing diagram of one embodiment of a DMA acknowledge read with one wait state present.

Figure 10 is a circuit diagram of one embodiment of a cross section of bus showing pipeline from master to a sample slave and back to master.

Figure 11 is a circuit diagram of one embodiment of a bus pipeline registers.

Figure 12 is a circuit diagram of one embodiment of a bus write pipe wait staging with bypass.

Figure 13 is a circuit diagram of one embodiment of a dascade and tree OR bus.

Figure 14 is a block diagram of one embodiment of a split DMA with simultaneous grant.

Figure 15 is a block diagram of one embodiment of a DMA selectors.

Figure 16 is a block diagram of one embodiment of a selector channeling.

Figure 35 is a block diagram of one embodiment of an address mapper.

Figure 36 is a block diagram of one embodiment of exporting of special function registers of 8032.

Figure 37 is a block diagram of one embodiment of bypassing a CSI bus.

Figure 38 is a block diagram of one embodiment of a Memory Interface Unit (MIU).

Figure 39 is a block diagram of one embodiment of a multiple clock source.

Figure 40 is a block diagram of one embodiment of a multi-CPSU system showing shared vs. unique signals with multiple slaves.

Figure 41 is a block diagram of one embodiment of a multi-CPSU system showing internal blocks of master handling slave interface.

Figure 42 is a block diagram of one embodiment of a JTAG as bus master.

Drawing convention: in the figures a white square is used to indicate one or more signals controlled by configuration memory that is configured prior to normal user operation of the programmable logic.

Detailed Description

1. Introduction

A configurable processor system unit (CPSU) integrates a CPU, an internal system bus, programmable logic also referred to as configurable system logic (CSL), and various system resources all interconnected and communicating via the internal system bus on a single piece of silicon. For one embodiment, one of the system resources provides a dedicated external memory interface function.

A typical application environment is shown in Figure 1. Here the CPSU chip 110 connects directly to a read only memory (ROM) 130 from which it accesses various programmed information to determine its operation. To control, monitor or communicate with its environment, other connections are via the input output

VCC	Supply voltage
VSYS	System initialization mode
RST_	Reset input (low assertion)
XTAL1	Oscillator input
XTAL2	Output of oscillator or primary system clock input

A block diagram of the CPSU 110 identifying one embodiment of the system resources is shown in Figure 2. Those system resources that reside on the internal system bus 225 are shown to the left of the bus and arbiter block 225 if they are masters, and to the right if they are slaves, in typical application. The CPU 220, CSL 210, PIOs 215, and memory interface 260 mentioned above are shown. The various system resources may also include internal random access memory (RAM) 255, a small internal ROM 250 containing fixed initialization code, a controller for performing direct memory access (DMA) 230, a JTAG interface 240 (Joint Test Action Group) with bus master extensions, and an internal bus monitor and breakpoint unit 245. The clock control 280 and power and reset control 275 sections generally do not have general bus interfaces. There is also a group of control registers that are not shown in the diagram. For one embodiment, these control registers determine the operating characteristics of each of the system resources. The memory bits that determine the configuration of the CSL 210 are assigned to an addressable region on the internal system bus 225. The CSL 210 also has the slave side of the internal system bus 225 distributed throughout along with many programmable bus interface decoders and pipeline registers that may then be used by the user programmable logic within the CSL 210.

2. Configurable System Interconnect

2.1. Overview

The programmable logic section referred to as configurable system logic (CSL) is connect d to the processor, various system peripherals, external pins, and

2.1.1. Primary CSI Signal Name Lists

All of the interconnect signals have a primary name listed here. Each of these primary signal names may be reused with different prefixes. The prefixes will typically resolve which name to associate with a particular pipeline time slot and bus segment. Signal names may also have aliases. The un-prefixed primary name will typically be considered to alias with the names of the signals available within the CSL at CSI Sockets. The prefix conventions will be explained after the primary name lists.

Common Bus Signals

These are the shared or common CSI Bus signals that are also available under aliases at every CSI socket.

bclk	Bus clock
dws[7:0]	Slave write data bus, into slaves
drs[7:0]	Slave read data bus, out of slaves
adr[31:0]	Address bus
waited	Indicates previous cycle was a wait state
waitnext	Assert to cause next cycle to be a wait state

Bus Control Signals

The following signals are not directly available inside the CSL at a CSI Socket. They do appear on the CSI Bus and are available for dedicated system logic. The signals are available during the pipeline distribution phase and are used as qualifiers in the selectors within the CSL.

brst_	Bus reset (not available at CSI Sockets)
dwm	Master write data bus (not inside CSL)
drm	Master read data bus (not inside CSL)

Bus DMA Signals

For one embodiment, there are two direct memory access (DMA) channels. The index identifies which channel the signals are associated with. Inside the CSL, the DMA signals are connected to the selectors.

dmareq[1:0]	DMA requests
dmaack[1:0]	DMA acknowledges

Breakpoint Signals

The breakpoint generator can receive a break signal from the CSL or have the final break delivered to the CSL.

bpevt	Event from breakpoint generator to CSL
bpctl	Control from CSL to breakpoint generator

All of the signals described above are generally synchronous to the bus. The signal groups related to bus operations are also used in the dedicated system logic. Within the CSL there are many copies, or entry and exit points for these signals, all of which are logically equivalent.

Bus Arbitration Signals

Arbitration signals are related only to the operation of bus master devices. Only slave devices may be implemented within the CSL. The portions of the bus that relate to master interface are implemented within dedicated logic. The index identifies which arbitration channel the signals are associated with. These signals are bus synchronous.

nw[7:0]	Need (request) write data bus
na[7:0]	Need (request) address and control bus
nr[7:0]	Need (request) read data bus

2.1.2. Basic Bus Operational Description

For one embodiment, the bus is 32 bit address, 8 bit data, non-multiplexed with separate data in and out, and multiple separate decoded and qualified read and write enables. All bus information is valid at the rising edge of the clock. The full cycle time is available for CSL logic, except for the output delay and input setup time of registers at the socket interface. The same CSI bus is used for both CSL and dedicated system logic. DMA requests may be pipelined and are pended to achieve burst operation. Each cycle on which a request is asserted is recognized as a separate request. A round robin arbitration scheme is implemented for masters; however, the CSL only supports slaves.

Bus Segments

A block diagram of one embodiment of the various bus segments labeled according to their name prefix is shown in Figure 3. The "arb_" arbitration segment 315 connects the arbitration request and response signals between the masters 320 and the arbiter 310. For one embodiment, the arbitration signals are separate for each master 320 and are not bussed. The "mw_" master-write segment 325 carries signals from the masters 320 to the bus pipe 330. The individual master signals are collected together via either multiplexors or logical OR gates into a consolidated bus in this segment. The "sw_" slave-write segment 335 buffers and distributes signals from the bus pipe 330 to the selectors and pipe registers 340 distributed throughout the CSL. The "fw_" socket interface write segment 345 carries signals from the selectors and pipe registers 340 to the slaves 350. Here the signals are routed to the general-purpose configurable logic of the CSL. The "fr_" socket interface read segment 355 collects signals from the slaves 350 to the return pipe registers 340. These signals are also routed through general-purpose configurable logic of the CSL. These "fr_" signals are collected together via either multiplexor cascades simulating logical OR gates or logical OR gates into more consolidated bus signals at the pipe registers. The "sr_" slave read segment 360 carries signals from the return pipe registers 340 located throughout the CSL to the bus pipe 330. Th y are collected via logical OR gates within this segment into a

segment. These transaction scenarios differ in the remainder of the time slots. Each of these is described separately below.

In the scenario shown in Figure 4, an addressed write with no wait states, the data is registered into the slave at the end of time slot "Time[N+2]" and the transaction is complete.

In the scenario shown in Figure 8, a DMA acknowledge write with no wait states present, the data is registered into the slave at the end of time slot "Time[N+2]" and the transaction is complete. The DMA acknowledge signal is owned and driven by the particular DMA channel making the transaction.

In the scenario shown in Figure 6, an addressed read with no wait states, th data is gated out to the bus on the "fr_" segment during time slot "Time[N+2]". This data is then piped to appear on bus segment "sr_" during time slot "Time[N+3]". The read data is consolidated and passed on to segment "mr_" during the same time slot. At the end of this time slot it may be registered back into the master that originated this transaction.

In the scenarios shown in Figure 5, Figure 7, and Figure 9, the addition of one or two wait states preserve the information on the "fw_" bus segment for one or two additional cycles respectively. In the scenarios of Figure 5 and Figure 7 the first wait state is added by the selector decode resulting in the assertion of "sr_waitnow" during "Time[N+2]". In the scenario shown in Figure 9 the wait state is inserted by something else on the bus, such as a simultaneous addressed read transaction like that shown in Figure 6, not by the selector handling the DMA. Whenever "waitnow" is asserted on the "sr_" bus segment, it is driven back to the "sw_" bus segment during the same clock cycle or time slot. Whenever "waitnow" is asserted, it results in "waited" being asserted during the next time slot.

In the scenario shown in Figure 5, an addressed write with one wait state, the data may be registered into the slave at the end of either time slot "Time[N+2]" or "Time[N+3]" or both. At that time the transaction is complete. If it is important

through the figure between each time slot and through the right edge of the registers assigned to separate those time slots. The right hand edge of each register in the logic diagram is drawn with a thicker line that is aligned with the applicable time slot boundary.

Within the diagram, the segments that collect or consolidate signals have a logical OR gate containing a dot in the center drawn on the applicable signals. This represents that the many sources or aliases for that signal are combined via logical OR gates. This symbol as used here does not imply any particular method or circuit for of implementing a logical OR. A circle at the point where a signal enters a block implies that the signal is inverted relative to its use within the block. The enable controlled registers are shown with an "e" control input that, if asserted, allows new data to be registered when it is clocked. The gated registers, shown with a "g" control input, may be implemented either as a latch or as a clocked register with a bypass multiplexor where the output of the multiplexor is the input to the register. Clock connections are not shown in the diagram; however, all of the registers shown are driven by the bus clock.

The signals have one clock cycle to travel from the outputs of a master 320A through the "mw_" master write collection segment, through the bus pipe 330, through the "sw_" slave write distribution segment, through the configure selector decoder 1040, and to the inputs of the "fw_" socket interface write registers. For more details on the configure selector decoder 1040, see Section 3.2.2.

The control and information on the "fw_" socket interface write distribution segment signals are then available for one or more clock cycle within the general purpose configurable logic. The selector may be configured to insert the first wait state. If more than one wait state is needed, the "fr_waitnext" signal is asserted during the first clock cycle within configurable logic.

If the transaction is a read, then the applicable "fw_" signals are used to enable information onto the "fr_drs" signals. An addressed read without wait states r turns the information enabled by "fw_rdsel" through the "fw_drs" socket interface read coll ction segment to the inputs of the data return registers.

programmable logic while the current set is handled within the logic and transportation of the previous return information back form the programmable logic at the same time. The total data bandwidth may be tripled by this pipelining. The duplication of the pipeline registers within the programmable logic minimizes the time to or from any pipeline register reducing the clock cycle time required within the programmable logic. It also, by minimizing the paths, simplifies the worst case timing analysis for bus interface timing. By leaving the bus transport time entirely outside the programmable logic the bus interface timing appears to be uniform at every point within the programmable logic.

All of the bus interface signals to the programmable logic passes through a set of registers before being available for use inside the programmable logic. All bus interface signals from the programmable logic pass through another set of registers before being returned to the requesting device. This separation of logic into multiple time domains is called pipelining.

Figure 11 shows how this may be accomplished. The bus "address", "write_data", and "controls" signal groups are distributed to the programmable logic 210 and then within that section are registered. After the registers 1110, 1130 two sets of these signals are shown as "inA" and "inB". The "read_data" signals generated within the programmable logic are registered by registers 1120, 1140 there, and then returned by the bus. Before the return registers 1120, 1140 they are shown as "outA" and "outB".

2.3. Write Pipe Wait Staging with Bypass

This write pipe wait staging with bypass allows a wait state to be inserted by the slave device on a pipelined bus and makes it appear similar to an un-pipelined interface at the slave level.

There are two alternate examples. In one case the bus is not pipelined and the wait is returned to the master which is responsible for holding the bus signals in their current state. In the second case, usually implemented with pipelined interfaces, the slave is responsible for capturing and holding any signal values it may need to complete the transaction. In the pipelined case, the signaling is

still meet the timing requirements, so the programmable logic is only allowed to make an advance request for a wait on a signal called "waitnextA" that is then registered an driven to the rest of the system. The initial cycle assertion of "waitnow" for a slave is generated by logic connected just ahead of the staging registers that feed "inA" since the programmable logic does not yet have information from which to do this.

2.4. Contention Free Bus

This method allows several sources to provide data to a common bus.

There are five alternate examples. In one case the data bus is a shared signal line that is actively driven either high or low by one source while not being driven by any other source. If inadvertently two sources drive opposite values at the same time, high current contention conditions may result. In another case multiplexors are used to gate each source into the data bus in two stages. Here the data bus passes through every multiplexor stage or an OR of the controls for the multiplexors in a subsection are used to multiplex it into a higher stage. In the third case, multiplexors are also used; however, they are all in a continuous ring such that prior to every multiplex block the same data can be picked up. The delay to any point is long but it does share the signals for write and read data. In the fourth case a wire OR bus is used for a back-plane data bus. In the fifth case in fully integrated designs ORs are used to combine data sources.

In this method a dedicated OR internal system data read bus is distributed throughout programmable logic. This provides a data read path for an internal system bus. It eliminates possible contention conditions of the shared signal bus. It also eliminates the composite controls of a two stage multiplexed bus. It improves the delay over a continuous ring. It is faster that a wire OR. As part of programmable logic it provides a higher performance system data bus than can be implemented with purely programmable logic yet does not require a new dedicated integrated design for each design application as with non-programmable logic.

from source to destination approaches one item transferred per clock cycle and a higher performance clock cycle is allowed.

A DMA transfer involves a read of data and then a write of that data. One of these, either the read or write, also requires an address and a read or write control while the other requires a DMA acknowledge control. The target of a particular DMA acknowledge will have already been set up to cause data transfer in the appropriate data direction.

In order to improve the performance of the data busses, the read data is separated from the write data, which means that the read data does not have to propagate back to all points on the write data bus in one cycle. This allows the cycle time to be reduced to near the time it takes to propagate data signals in one direction through the system rather than two directions.

With the data read bus split from the data write bus, the two parts of a DMA transfer use independent yet complementary bus signals. Therefore, they may be performed independently in a fully overlapped manner. The corresponding read and write transactions cannot be performed simultaneously since the data to be written is not available until after it is read. Since DMA is only used for larger blocks of data, data that is read may be saved in a buffer and then written simultaneously with another subsequent read.

For one embodiment, each DMA channel is implemented as two cooperating master devices. One of these will perform a series of reads, placing the data in a FIFO (First In First Out) buffer. The other will perform a series of writes taking the data from that same buffer. The master doing the writes, will only do so when there is data in the buffer. The master doing the reads, will only do so when there is space in the buffer to place the data and it has been requested to make the transfers. The buffer will have space for several transactions so that the read process can continue while the write process gets started. In order to access the bus each master arbitrates for bus resources.

The internal system bus is divided into three sets of signals that may be arbitrated for. The read and write data busses are separate but may be used

which of the masters 1410, 1420 will execute the read and when, and other typical DMA transfer characteristics.

2.6. DMA Channeling

This method allows many programmable decoders to be individually used to direct DMA requests and acknowledges within a programmable logic structure.

There are two alternate examples. In one case for there is a dedicated DMA channel and dedicated bus request and acknowledge lines for each device. In the second case the channels and request and acknowledge signals associated with a particular DMA device may be statically configured, however, several devices may share one channel each with its own dynamic enable.

In this method a DMA device may be dynamically configured to use any or none of the available channels so that the channels may be shared over time. It also allows reuse of signal paths into the programmable logic and reuse of the decoders providing flexible addressing of the control bits.

The internal system bus includes signals for DMA channels. One of these DMA channels may be selected at each DMA channel interface. These DMA channel interfaces are distributed throughout an array of programmable logic. At each interface block, control bits select which channel may be enabled and whether it is enabled. When a channel is enabled, requests are passed out from the programmable logic to the selected channel and acknowledges from the selected channel are passed into the programmable logic. The internal system bus interface also includes address and command decoders distributed throughout the programmable logic array in the same manner as the DMA channel interfaces so that some of their logic may be shared. The same signal path into the programmable logic may be used either for the decoded read and write signal from the decoders or as the DMA acknowledge signal. When the interface signal is used for DMA the address and command decode is reused to access the corresponding DMA channel selection and enable control bits. This allows the reference address for these two control bits to be placed anywhere within the system address space. Whether the interface element is to be used

2.7. External Memory Access

This method allows several programmable address and command decoders and a memory interface slave to cooperate in accessing space defined by the programmable decoders.

There are two alternate examples. In one case the external memory interface has dedicated decoders for any space it is used to access and dedicated decode outputs for each decoder. In the second case, each memory interface has another set of address, data and control signals.

This method reduces the number of additional external interface signals required to interface to additional external memory devices. The address, data, and read and write control signals of a single memory control interface slave may be used in conjunction with decoders distributed throughout some associated programmable logic. This also allows the added decode-signal to be routed to any of the programmable logic pins.

The system includes an internal system bus and a memory interface unit that may respond to that bus. The memory interface has signals to communicate with, and control an external memory device. The system also includes a section of programmable logic and programmable address and command decoders distributed throughout that logic. The bus and programmable decoders are extended to include an additional bus signal from any of the decoders that communicates to the memory interface unit. When this signal is asserted it causes the memory interface unit to perform an external memory transaction similar to one it would decode itself with the exception that it will not assert the standard chip enable signal. The external address, data, read and write signals of the memory interface unit are driven with the transaction information from the internal system bus. The chip enable for the decoder that requested the transaction may be connected through the programmable logic within which it resides via any available external interface pin to the selected memory device. A control register bit for each command decoder is used to determine if it will request that the memory interface unit perform the transaction. The programmable logic retains control of the overall timing of the

3. Configurable System Logic Matrix

In the following sections, the CSL matrix is described from two complementary points of view, namely the block level view and the signal-flow view.

3.1. Internal Structures

3.1.1. Overview

When integrating programmable logic with a CPU or other on-chip system-level components, it is necessary to provide mechanisms for connecting system-level resources to the CSL. System-level resources may include but are not limited to system data, address, and control signals, pipelining registers, buffering, and miscellaneous logic for interacting with system-level components such as the CPU.

It is difficult to achieve satisfactory system-level integration of a homogeneous CSL simply by connecting system-level signals to the edge of the CSL. This approach, while simple, fails in a number of respects for homogeneous CSLs. For example, it fails to prevent undesirable signal skew of system-level signals in different parts of the CSL.

To achieve satisfactory system-level integration of a homogeneous CSL, it is necessary for some of the system-level circuitry to co-reside with the programmable resources in the interior of the CSL. Different arrangements of system-level circuitry and programmable resources exhibit different combinations of benefits and costs.

This method establishes a hierarchical distribution of system-level circuitry with the programmable resources in a homogeneous CSL, in such a way as to gain the benefits of hierarchical organization while retaining the benefits of a homogeneous array.

A homogeneous CSL consists of a more or less regular array of similar units of programmable logic resources. These units are called tiles, because they correspond to design units of physical layout, which are then tiled to form an array.

Pipeline registers at each bank boundary exist for common timing for different sized die to simplify system level timing.

This method retains the benefits of a homogeneous organization, and allows the application of well understood placement and routing algorithms.

This structure brings system-level circuitry in close topological proximity to programmable logic throughout the CSL. The maximum physical distance between pipelining registers and any given programmable resource is limited by the size of the bank. By choosing an appropriate bank size, different cost/performance tradeoffs are possible. This structure increases the predictability of the timing of system-level signal paths in the CSL, which simplifies software algorithms and typically increases the overall system performance for designs implemented in the CSL. This structure scales very well with different array sizes. The average distance between system-level circuitry and programmable resources stays constant as array size grows. This method enables the programmable separation of the banks into relatively isolated resource pools that can be operated on independently by placement, routing, and other algorithms. This divides what is typically a large and difficult problem into several smaller and simpler problems. As a result, algorithm run times are significantly reduced.

This method reduces the burden that the tiles in the bank bear for supporting integration of system-level signals. By concentrating the system-level connections in the breaker tiles, the other tiles in the bank are relieved of the need to support high-performance integration with circuitry that is physically or topologically remote. This simplifies the design of the other tiles.

3.1.2. Two-dimensional Bank Organization

Figure 18 shows a block diagram of a 3x3 CSL matrix 210. A CSL matrix 210 can be built with any number of rows and any number of columns of logic banks. A logic bank 1810 contains user programmable logic and the necessary logic to distribute the CSI system level signals. Some forms of general

Eight address decoder outputs feed from a logic bank 1810 to the vertical breaker 1820 above, as an address selector is distributed in both parts. The selector outputs are then distributed to the logic bank below via the general interconnect.

A set of global signals is distributed similarly to the data write bus.

A copy of a set of control write signals goes down every column of horizontal breakers 1830 through the corner breaker 1840. A buffered version turns the corner into the vertical breaker 1820 and is distributed to the logic bank 1810 below through the general interconnect.

A set of control read signals is collected from a logic bank 1810 via the general interconnect, processed in the vertical breaker 1820 above and turns corner in the left corner breaker 1840, combining with the control signals from below before going up. Control signals from every column of corner breakers 1840 are then combined in the interface logic.

In order to make the CSL scalable and meet timing requirements, there is a 2-D wait resolution mesh mechanism to handle the system wait signal.

Although specific orientations are described above with respect to the routing of busses and signals, it is to be understood that alternative routing may be implemented and is within the scope of this description.

On the CPSU chip, as in most other chips, there is a ring of I/O pads 1910 on the perimeter. For one embodiment, a programmable I/O (PIO) pad 1910 has 3 wires connected to the internal core logic, namely the data input, data output, and output enable. Figure 19 shows one embodiment of how the pad ring 1910 is connected to the logic bank 1810 at the edges of the array. For details, see Figure 27.

Figure 21 shows more details of the logic bank 1810, vertical breaker 1820, and horizontal breaker 1830. In one embodiment, a logic bank 1810 consists of a 8x8 (row x column) array of logic block tiles 2110, a horizontal breaker 1830 consists of a 8x1 array of horizontal breaker tiles 2130, and a vertical breaker 1820 consists of a 1x8 array of vertical breaker tiles 2120, while a corner

to match any 32-bit address value, where each bit value is either a ONE, ZERO or DON'T CARE.

The data path 2220 contains an array of four data path elements 2260, with each element handling one bit of the data read bus. For example, data path element index 0 2260 receives its input from horizontal path signals die[0], diw[0], and generates output to horizontal path signals doe[0], dow[0]. Moreover, it generates output dout[0] to feed a OR gate 2270, which combines this signal with the incoming ifr_drs[0] signal to generate the outgoing ofr_drs[0] signal. For one embodiment, the OR gate 2270 is part of a chain that spans the entire column of a logic bank. In parallel is a four-bit data read bus sr_drs[3:0] that feeds through the entire logic bank and is combined with the registered version of ofr_drs[3:0] in the vertical breaker tile (see Section 3.1.5).

Each data path element 2260 can be configured to either inject a data value into both eastward and westward data injection muxes to drive signals doe and dow, or allow signal die to drive dow and signal diw to drive doe as if the injection muxes were a feed-through. The output signals of doe and dow are OR'ed together to generate an output signal "m". Signals m[3:0] feed the block of programmable logic and interconnect. These signals may then be transformed and/or transmitted in any fashion allowed by the specific implementation of the user configurable logic and interconnect.

The "m" signal is also AND'ed with a configuration element bit to enable its value to appear on the output dout. As the data read bus is an OR bus, this enable bit effectively blocks the "m" signal from affecting the data read bus value. As one of many possible implementations, the injected data value is selected from one of two possible sources d[1:0] feeding a 2-to-1 mux, and the common injection mux select signal is controlled by the output of a 4-to-1 mux with four signals including VCC, GND and two signals sel[1:0]. The data sources d[1:0] and select sources sel[1:0] are common to all 4 data path elements. They are output signals from the block of programmable logic and interconnect.

Within a horizontal connector 2310, there is a 4-to-1 mux 2350 selectable by asele. One of the four address input signals feeding the mux 2350 selectable by asele as output feeds a pipeline register enabled by inverted waitnow, and then conditionally drives the east long line lhe. The pipelined address injection muxes can also be configured such that either lhe drives lhw or lhw drives lhe so that lhe and lhw can effectively work as two segments of a horizontal long line spanning across the horizontal break tile.

Other possible designs of horizontal breaker may be used. See Section 3.1.8.

3.1.5. Vertical Breaker Tile

Figure 24 shows one embodiment of a vertical breaker tile consisting of circuits dealing with the write data bus, the read data bus, the control busses and an array of 8 vertical connectors 2410.

For one embodiment, there is a set of four pipeline registers 2430 enabled by inverted waitnow for the 4-bit subset of the write data bus. The pipelined write data bus output signals are the only write data bus signals accessible to user programmable logic within CSL.

The 4-bit subset of the data read bus coming from the logic block tile below is fed as data to a set of 4 pipeline registers 2440. Each registered output can be selected as the output of any of the four 4-to-1 muxes 2450, where the output of such a mux 2450 is OR'ed 2460 together with the corresponding incoming pipelined read data bus. The purpose of such muxes 2450 is to allow arbitrary 4-way permutation of the assignment of read data bus signals to facilitate placement and routing of user programmable logic. Note that the corresponding OR gates 2460 for bits 1 and 0 are 3-input OR gates, where the additional inputs are outputs cfr_drs[1:0] from the bus state transformer (See Figure 25).

Each vertical connector 2410 can be configured to either allow lvn to drive lvs, lvs to drive lvn, or inject a data value from the output of a 4-to-1 mux. As shown in Figure 24, the four source signals common to all 8 vertical connector data injection muxes are fw_wrsel, fw_rdsel, fw_waited and fw_bpevt. For one embodiment, while fw_wrsel and fw_rdsel feed all muxes, fw_waited feeds only

signal owait is generated. The owait signal is then distributed to the south feeding the horizontal breaker tile 2130, and also to the east feeding the vertical breaker tile 2120, and the entire lower right logic bank 1810 via the vertical connector logic to the vertical long line and to the user programmable logic.

The owait signal is the OR output of 5 input signals, namely the local isr_waitnow and the wait input from four corner directions (waitnow_issw, waitnow_inne, waitnow_iwnw, waitnow_iese). These four signals are respectively the OR combined wait signals from nodes in the north-west, south-east, north-east and south-west regions. The other four OR gate outputs are respectively OR combined wait signals to the north-west, south-east, north-east and south-west regions.

For another embodiment, the same functionality may be implemented by snaking a 1-D OR-bus in a row-major or column-major fashion with a longer delay.

3.1.7. Programmable I/O Tile

As shown in Figure 27 is a programmable I/O tile 2710. At this level of organization, it looks very like a logic block tile with the following exceptions. There is no address decoder, but there is a set of signals connected to the corresponding I/O pads 2760. For each programmable I/O cell 2740 inside the block 2720, there are 3 signals connected to the corresponding I/O pad 2760, namely di, do and oe.

3.1.8. Alternative Logic Block Tile and I/O Block Tile

Figure 28 shows an alternative logic block title 2810. It is more complicated than the one in Figure 22 due to additional interconnect lines between adjacent blocks. In this illustration there are 3 outputs to the north, 1 output to the south, 3 outputs to the west, 3 outputs to the east, and 1 output to the south-east 2850, 1 input from the north, 3 inputs from the south, 3 inputs from the west and 3 inputs from the east.

Similar addition to interconnect can be added to the I/O block tile.

vertical short lines (svn[7:0], svs[7:0])), and 10 input signals (the 6 block outputs from neighboring logic blocks and 4 outputs (m[3:0]) from local data path muxes.

3.1.9. Zip Switch Box

Figure 31 shows the details of a zip switch box 3010. It is named a zip switch box 3010 because of the existence of the zip lines. A zip line is an input mux followed by a buffer followed by one or more output connections as part of the zip switch box. There are three sub-blocks, namely, ziph 3110, zipv 3130, and sbox 3120.

The ziph block 3110 contains an array of 8 horizontal zip lines 3140. Each horizontal zip line 3140 can select its inputs from one or many signals. For one embodiment, each horizontal zip line 3140 can select from 30 signals (lv[7:0], svs[7:0], din[3:0], and the 10 block outputs and datamux outputs). Each zip buffer output may be configured to drive or not to drive its corresponding horizontal long line. The outputs of the ziph block 3110 drive the sbox 3120.

The zipv block 3130 contains an array of 8 vertical zip lines 3150. Each vertical zip line 3150 can select its inputs from one or many signals. For one embodiment, each vertical zip line 3150 can select from 30 signals (lv[7:0], svs[7:0], global[3:0], and the 10 block outputs and datamux outputs). Each zip buffer output may be configured to drive or not to drive its corresponding vertical long line. The outputs of the zipv block 3130 a drive the sbox 3120.

As shown in Figure 32, the sbox 3120 contains an array of 8 simple ten-way switches 3210. Each such switch 3210 performs switching among a plurality of signals. For one embodiment, the sbox 3120 switches among the following 8 signals in 10 different ways. For switch index k, the input signals are respectively shw[k], she[k], svn[k], svs[k], zh[k], zh[(k+1) mod 8], zv[k], and zv[(k+1) mod 8]. These signals correspond to the ten-way switch signals w, e, n, s, zhw, zhe, zvn, and zvs respectively. The 10 possible programmable connections are w-e, n-s, n-e, e-s, s-w, w-n, zhw-w, zhe-e, zvn-n, and zvs-s. Two implementations are shown 3220, 3230. They both use 10 transistors, the

restriction, these 8 bits all come from the same position of each of the 8 address bit bundles. As all eight 4-to-1 data injection muxes share the same two configuration memory cells, there are only four choices, namely a[31:24], a[23:16], a[15:8] and a[7:0]. This is exactly why the address bundling is defined as such as it is more usual to use consecutive address bits. Note that for each horizontal long line, the corresponding horizontal connector can be configured to allow injection of the address bit or not.

As addressing braiding does not change the 4-bit bundling definition, in the next row of logic tile, address bits a[31:24], a[23:16], a[15:8] and a[7:0] are still the four possible choices. However, due to braiding, the exact long line where an address bit appears changes. For example, if a[31] is available on lh[7] of a horizontal breaker tile, then a[31] will appear on lh[6] of the horizontal breaker tile below.

For one embodiment, because the address braiding scheme has a cycle of 8 and there is no braiding in the corner breaker tile, the whole pattern repeats itself for every logic bank. This is advantageous for placement and routing of user programmable logic.

As a possible enhancement, it is possible to use a modified horizontal breaker tile 2910 that drives to both left and right tiles as shown in Figure 29, so that the column of horizontal breakers on the left edge of CSL can be degenerated such that all the address buffers and horizontal connectors can be eliminated.

3.2.2. Address Selectors

For one embodiment, there is one address selector per bank column so that there are 8 address selectors per bank. As shown in Figure 21 and Figure 22, the 32-bit address decoder of each bank column is implemented as a cascade chain of 8 four-bit address decoders, one per logic block tile. The last decoder output of the chain is then fed to the north-neighboring vertical breaker tile (see Figure 24), entering the CSI control bus transformer box (see Figure 25), combined with other CSI bus signals, generating output signals fw_rdsel and fw_wrsel. These signals are then conditionally injected into the general

along the same row, appearing as dout on every data path element along the chain. At the column where the target bit position is located, the dout signal is enabled to be OR'ed to the unregistered version of the data read bus. In fact, the signal can be injected to any one of the four chains locally and will still be able to get to the correct bit position via the set of four 4-to-1 muxes in the vertical breaker tile.

3.2.5. Data path Mux Chains

For each logic bank, there are 32 pairs of data path mux chains running horizontally, optionally connected to their corresponding ones in adjacent logic banks via the data path muxes in neighboring horizontal breakers (see Figure 23). Along each row of logic block tiles, there are 4 pairs of data path mux chains, with each pair corresponding to one bus bit.

In normal operation as a bus, only one active signal is dynamically injected to a pair of data path muxes, with the output propagating to both east and west, while the other pairs of muxes on the same chain are dynamically selected to be in feed-through mode to allow that signal to propagate. At each data path element, the outputs of the two muxes are OR'ed together to generate the equivalent correct bus signal, whether the active signal source is from the east or the west.

The length of the chain can be chosen very flexibly. As the active value is a logic 1, a logic 0 (GND) signal is used at inputs at the two end points of a chain for proper operation (see Figure 19). However, any given data path element can be configured to always inject (with the output of the injection mux select mux tied to VCC) a logic 0 to both injection muxes so that it behaves like the simpler data path element of the horizontal breaker tile (see Figure 23). Moreover, if a data path element along a data mux chain chooses not to participate, it can always be configured to be in feed-through mode.

The data path mux chains not only serve as an extension mechanism to collect system data read bus signals, but also can be used in a stand-alone manner as

a user bus, as long as the dout signals are disabled to not affect the system bus.

Since the configuration bits for the 4 pairs of data path elements are not shared, it is possible to configure CSL to support multiple busses of differing widths.

As another mode of operation, the data path mux chain can be configured to serve as auxiliary general interconnect, as the "m" output signals are output from the chain to the programmable logic and interconnect block, and thus entering general programmable interconnect. In the specific case of Figure 30 and Figure 31, the "m" signals enter the zip switch box as inputs.

The mux chain pair can also be used as logic, each half of a pair as a priority mux chain. The priority of an injected signal is determined by the relative position of the data path element on the chain. For example, the east propagating priority mux chain has it highest priority element on its east end. For one embodiment, two parallel priority mux chains are operated in parallel, with the east end "m" output and the west end "m" output as the respective final output with two opposite priority ordering.

The priority mux chain needs initialization at both ends. Data path muxes along the chain not participating can also be set to bypass mode.

Those skilled in the art understand that one may replace the OR gates in the mux chain by AND gates, and GND signals at the two ends by VCC signals, to achieve the equivalent functionality through De Morgan's Law.

3.2.6. Control Signals

There are 3 groups of controls signals as classified by their directionality. They are the 6 control write signals (sw_rden, sw_wren, sw_dmaack[1:0], sw_xmact, sw_bpevt), the 4 control read signals (sr_dmareq[1:0], sr_xmreq, sr_bpctl), and the wait resolution signals including sw_waitnow, bsr_waitnow. For details of the wait resolution, see section 3.2.7.

The control write bus is buffered at the interface logic at the top of CSL to generate a copy for every column of corner breakers (s e Figure 19). As the

to be examined throughout the system. A large wired OR is typically very slow or requires large amounts of power.

In this method the wait control signal is propagated in an array structure that allows a source to drive it at any point in the array and provides a logical OR of all sources to the array at any point in the array. This wait mesh provides this combined wait result with a minimal worst-case propagation path that only traverses the array in one direction. All of the signals and gates involved may be relatively small and low power. For one embodiment, the logical element and interconnect used here to propagate the wait control signal is identical at each point in the array.

For one embodiment, the system logic that handles the wait signal is physically structured as a two-dimensional array where each element of the array is repeated in vertical columns and horizontal rows with vertical and horizontal connections between the elements. At each of these array elements a local wait control request signal may be driven into the element. The composite wait request result of this and all of the corresponding inputs at the other elements of the array is output from this array element. The logical structure local to each of these elements is identical. The logic within each element combines the local wait request signal with directional wait request signals traversing through the array element via logical ORs so that it is sent out in all directions. The directional wait request signals from all directions are also combined within the array element and with the local request via a logical OR and delivered out for local use.

The directional propagation signals that traverse the array between the array elements and the logic within those elements to facilitate that traversal are chosen such that each local request may not be returned to the original array element at which it entered yet is propagated to all other elements. A set of eight directions are used to propagate the wait control. Two signals physically drive out to each of the adjacent column and row array elements thus driving a total of eight directional signals. Similarly two signals are input from each of the adjacent column and row elements. Each of these input signals is propagated

with an 8032 CPU, it is sufficient to connect sideband signals just to the unused imuxh outputs and unused inputs of the zip switch box of the logic tiles on the top edge. There is a total of 24 lines out of and 8 lines into CSL per bank, so that incoming sideband signals have more than one way of coming in, and outgoing sideband signals have more than one way of going out of the CSL.

3.3. CSL Configuration

3.3.1. CSL Configuration Memory as System Addressable Memory

The configuration memory space is randomly addressable as part of CSI system address space. Hence, it does not require any dedicated hardware to perform CSL configuration. Configuration process becomes an infinitely flexible software process in which the on-chip CPU simply writes the configuration data to the appropriate configuration memory locations.

For one embodiment, the configuration memory is structured similarly to a conventional SRAM memory array with word and bit lines. To save silicon area, the system data write and system data read bus is reused for the purpose of CSL configuration.

In case the configuration memory is unused for configuration, it could be reused as general purpose RAM to increase on-chip RAM capacity. Depending on the particular implementation, some additional constraints may apply.

3.3.2. Partial Reconfiguration of CSL

If not properly designed, during the CSL configuration process some wires may float and cause a high current condition and the chip may fail. In order to prevent this undesirable situation, there is a global enzip signal that controls the state of such wires. At power up, the processor writes a '0' to some control register location, which is memory mapped in the CSI address space, causing the globally distributed signal enzip to be false, and thus forcing all relevant lines to GND. After CSL configuration is completed, the processor writes a '1' to the same control register location, before normal CSL user mode begins.

4.1. Holding a bus cycle

This method adapts an 8032 type MCU, originally designed to operate as a solitary master with fixed response time, to work in a multi-master shared bus that has a "wait signal controlled" variable response time.

There are several alternate methods for achieving multi-mastership. One is to alternately allocate a machine cycle of a fixed number of clock cycles (twelve cycles in the case of a standard 8032 MCU) to a DMA controller, which is usually entirely integrated with the MCU. In another case, the MCU includes a control register that allows the number of four-clock machine cycles for external memory references to be specified. One embodiment described below provides a bus wait signal that is used to delay the generation of a bus cycle completion signal to the 8032's internal state machine, thus allowing for a variable number of clocks to complete a bus cycle.

This allows the 8032 to share a bus with several other bus master devices and to work efficiently with external slave devices of various speeds.

The internal state of the MCU machine cycle is monitored to determine when a bus command should be issued. When the clock cycle on which the new command should be issued is reached, if the bus can not yet issue the command the internal state of the MCU is held static until the clock cycle on which the command may be issued. The point at which a command may be issued may then be delayed. An arbitrator may then be introduced to control which of any number of devices will be allowed to issue the next command on the bus.

The internal state of the MCU machine cycle is monitored to determine when a response is needed. When the clock cycle on which a response is needed to proceed is reached, if the response data is not yet available the internal state of the micro-controller is held static until the clock cycle on which it does become available. The bus responses may be delayed an arbitrary number of clock cycles by each device responding yet no extra cycles are wasted for any particular response.

Three different source address spaces of an 8032 type MCU are translated by this method. The code and external data addresses, each a 16-bit space, and the direct registers, an 8-bit space. The source address is decoded in parallel for several source translation zones, eight for one embodiment.

Each decode is executed by comparing the high order eight bits to a value set for each zone, masking off any lower order comparisons according to a compare size set for that zone, and combining the result of the individual bit comparisons, masks and an enable for each zone.

The decode results are then handled in a priority order such that the address replacement defined for the highest priority decode will be the only on performed. The address replacement is done by substituting just the address bits used for the decode described with a destination address and further appending higher order destination address bits. The low order address bits within the zone that are not involved in the decode are passed through unchanged. The decode and replacement address bits as well as the size and enable may be either fixed or are user defined values. Two alternative bytes are used for the high eight address bits of code address depending on whether the address is an operation code or an operand, which are the first byte and successive bytes of an instruction. This allows these bits to be used to provide additional transaction type information on the address lines when used with suitable destination decoders.

Figure 35 shows one embodiment of address and command translation for the code and data addresses. The "cpu_adr" signals provide the source address while the "inst" signal indicates whether the source address is from code space or from data space. The "opc" signal further indicates when it is code whether it is an operation code if asserted versus an operand if not. The "crbits" signal collectively indicates all control values including the decode zone size and location, enables and the target address replacement values. The "trans_adr" signal is the final translated address while the "trans_adr_in" and "trans_adr_out" are the successive translation results from each stage.

bus cycle is generated outside the MCU to either write or read the corresponding externally defined register. The internal transaction is blocked. The state of the MCU is held at the current state whenever the external reference has not been completed. The internal write data is provided to the external bus and the external read data is provided to the internal bus.

Figure 36 shows one embodiment of register exportation. The added "direct_reg_read" 3630 and "direct_reg_write" 3620 signals are monitored to indicate that a read or write of a direct register is being made. The added "direct_reg_address" 3640 signal is monitored to indicate which register is being referenced. For one embodiment, an export control table 3610 implements an area where in some other address space the user may write and read the values and it is read only, one bit at a time in the direct register address space. For one embodiment, the export control table 3610 is implemented as a dual-port RAM.

This table 3610 may also be implemented as a ROM in the direct register space, in another logical form, or in some combination of forms. The added "direct_reg_export" signal indicates that either a direct register read or write is taking place and that the addressed register is to be exported. The added "direct_reg_write_data" and "direct_reg_read_data" signals are used to transfer the register data on a write or a read respectively. All of these added signals are then used to generate an appropriate bus transaction outside the MCU 220. The "direct_reg_export" signal is also used inside the MCU 220 to control the local source and destination of the register data. The added "hold" signal 3650 is used to keep the MCU 220 in its current state while it waits for the transaction to proceed.

All of the signals of the four ports 3660, shown in Figure 36 as 8-bit bus signals "p0out", "p0in", "p1out", "p1in", "p2out", "p2in", "p3out", and "p3in" may be eliminated and their interface functionality recreated within logic connected via the "external_bus_interface".

system bus. In the case where only one master is present, the issue of ordering disappears, and the master device need not arbitrate for the pipeline bus when it uses the bypass bus.

5. Other CPSU On-Chip Resources

A Configurable Processor System Unit, CPSU is a device that integrates a multitude of functions and technologies on a monolithic integrated circuit device. At least one or more of the following type of functions are integrated on a single silicon substrate to constitute a CPSU device: a central processing unit, an interconnect bus, and configurable system logic, or CSL, commonly known as Programmable Logic. This section describes several categories of integrated on-chip functions that improve CPSU's usability and performance.

Adding on-chip volatile or non-volatile memory provides the microprocessor an option to execute code from internal memory, hence reducing power and increasing performance. On-chip non-volatile memory in the form of PROM, or other erasable technologies, is optionally integrated to provide a pre-canned boot-up algorithm that manages the configuration of the CSL matrix and initialization of CPSU system registers.

A CPSU device with an integrated central processing unit, an interconnect bus, and configurable system logic is very flexible and can be configured to provide an external memory interface by simply programming the appropriate logic into the CSL matrix. The pattern for customizing the CSL, referred to as a bit-stream, usually resides in an external non-volatile memory device connected to the CPSU. This represents a problem since the bit-stream for establishing an external memory interface resides in the same external device that the CPSU will be connected to. An on-chip memory interface unit, MIU, overcomes this problem while providing several other services that are described below.

As systems get more and more integrated on a single silicon substrate debugging becomes increasingly difficult due to inaccessible pins. For one embodiment in-circuit debugger hardware is incorporated into the CPSU. The

The non-volatile memory resides on the system bus as described in Figure 2, and for one embodiment is located at system address Hex 0000. Upon exiting the Power-on-reset state most microprocessors including the 8032 MCU start executing code from memory at address location Hex 0000. The binary machine code that resides in this memory establishes the unique requirements that initialize the CPSU and prepare it for the application program. This is also known as Boot code.

It is not necessary to have on-chip non-volatile memory in the case where an off-chip non-volatile memory exists. However, in some applications, whether it is for security, for lower cost, or for field upgrade-ability, the on-chip non-volatile memory provides significant system advantages.

5.3. External Memory Interface Unit

An external memory interface provides many important functions for the CPSU. The pattern of 1's and 0's for customizing the CSL as well as the CPU program usually resides in an external non-volatile memory device connected to the CPSU. The memory interface unit provides a connection between on-chip CSI system bus and external serial or parallel byte wide memory.

In the case where the amount logic in one CPSU device's CSL matrix is insufficient, provision for multi-CPSU logic expansion is made. The expansion port is the memory interface unit operating in a single master, multi-slave environment.

Additionally, significant manufacturing test time reductions are made possible through a test mode operation in the memory interface unit.

5.3.1. Serial and Parallel External Memory Interface

A method is described that enables a system memory interface to communicate with parallel as well as serial static memories. It allows a system to interface to different types of external static memories, parallel or serial with the same controller.

CE- drives the chip-enable input of the external serial PROM. This bit is set by a Power-On or System Reset event.

SDOUT is the external serial memory data. This bit is only used when programming an in-system programmable, external, serial PROM. This bit is unaffected by a reset.

SDOUT_EN- is the external serial memory write data output enable. In serial write mode, this bit provides direct control over the three-state enable line of the output data buffers. This bit is cleared by a reset.

SDIN is the serial memory read data. In serial write mode only, this bit corresponds to D0 of the external data bus. It enables software to read the acknowledge status while programming the serial memory. This bit is read-only.

5.3.2. Multi-CPSU Systems

A multi-CPSU system provides CSL logic expansion when the logic capacity inside a single CPSU device is insufficient for a target application.

In a multi-CPSU system a single master device programs the slave devices. A dedicated pin 3910 (slave pin) indicates whether or not the CPSU 110 operates as a master or slave device as shown in Figure 39. When this pin 3910 is set, slave operation is enabled and a source selector 3940 selects the external crystal's 3920 CLK-IN pad as the clock source. The master device, for one embodiment, begins operation using its internal ring oscillator 3930 upon powering on and switches to the external crystal's 3920 CLK-IN source before it begins configuring its own CSL matrix. After the master device configuration completes, all remaining slaves connected to the master's memory interface unit are programmed.

The slave pin 3910 is also used to keep the CPU of each slave chip in reset until they are completely configured. Once configured, the slave chip CPUs are released by the master chip through the use of one of the system control

In a multi-chip environment, th master chip 4010 communicates with the slave chip 4020 through its standard memory interface. Using the same address, data and control lines 4130, the master chip 4010 can generate read or write transactions targeted to one of the slave chips 4020. Each slave chip 4020 uses a dedicated chip select pin 4140. These changes are equivalent to adding an external memory. Therefore, the same mechanism used to extend the external memory sub-system, described above, can be used in a multi-chip system.

5.4. System Debugger

5.4.1. JTAG as CSI Master

This method adapts a boundary scan JTAG port that was originally designed to test chip boundaries to operate as a master on the shared internal system bus and have access to all system resources for efficient system debugging. Figure 42 shows the JTAG as a bus master.

A multi-master / multi slave internal system bus 4290 connects to several building blocks including a CPU 220, DMA unit 230, memory devices 4230, and peripherals. The JTAG unit 4240, as a bus master, is capable of requesting a bus access. When the JTAG unit 4240 gains control over the internal bus 4290, it is capable of accessing any addressable element connected on the system.

A computer 4220 acting as an external tester or debugger can interface with the CPSU device 110 via its JTAG port, using a cable connected between the computer and the target board as shown in Figure 42.

The JTAG unit 4240 can access all addressable system resources by becoming the bus master on the internal CSI bus 4290. Serving as a bus master, the JTAG unit 4240, for one embodiment, converts serial bit streams into parallel registers whose contents are placed on the address, data and command buses to emulate CSI bus transactions. A JTAG unit 4240 also directly accesses internal MCU registers and CSL configuration data with visibility to functions not accessible from application programs.

During a CPU freeze period, the JTAG unit 4240 can poll any addressable location residing inside or outside the CPSU 110 and send all requested information to the computer 4220 for further display and analysis.

At the end of the debugging session, JTAG unit 4240 clears the breakpoint "freeze MCU" condition and the 8032 microcontroller 220 resumes code execution from where it left off. Alternatively, the JTAG unit 4240 can restart code execution from memory location 0000h by issuing a J_RESET command 4250 to the MCU 220.

At any point in time, the JTAG unit 4240 can instruct the MCU 220 to enter a single-step operation and send pertinent display information to the computer 4220.

The JTAG port can also be used to initialize the CPSU 110 or to update external memory devices connected to the MIU port. Using external Flash memory 4210, the JTAG unit 4240 can download a Flash-programming algorithm to the CPSU's internal RAM. The external Flash memory 4210 may include initialization data 4213 and/or user code 4214. The JTAG unit 4240 can interact with the internal MCU 220, allowing the processor 220 to control the actual program / erase / verify algorithms while the JTAG port supplies new data for programming or a new series of commands to the MCU 220.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

8. The configurable processor system unit of claim 7, wherein the processor has smaller separate address spaces for code and data, relative to a larger single system address space.

9. The configurable processor system unit of claim 8, wherein the smaller separate address spaces comprise 16-bit spaces.

10. The configurable processor system unit of claim 8, wherein the larger single system address space is a 32-bit address space.

11. The configurable processor system unit of claim 1, further comprising a random address memory (RAM).

12. The configurable processor system unit of claim 1, further comprising a read only memory (ROM) or a non-volatile type memory.

13. The configurable processor system unit of claim 12, wherein the processor and the programmable logic are bootstrapped from a boot portion of the ROM.

14. The configurable processor system unit of claim 13, wherein the processor and the programmable logic are further personalized as part of the bootstrapping.

15. The configurable processor system unit of claim 14, wherein the further personalization is part of a subroutine call by the boot portion of the ROM to an external memory.

16. The configurable processor system unit of claim 1, further comprising an external memory interface unit,

26. The configurable processor system unit of claim 1, wherein programmable logic is a bus slave.

27. The configurable processor system unit of claim 1, wherein the at least one region of programmable logic is configured by the processor.

28. The configurable processor system unit of claim 27, wherein configuration memory cells of the at least one region of programmable logic are addressed as system random access memory (RAM).

29. The configurable processor system unit of claim 28, wherein a configuration information to configure the programmable logic is retrieved from an external memory.

30. The configurable processor system unit of claim 28, further comprising a memory interface unit and a bypass bus, the memory interface unit coupled to the processor via the bypass bus, the memory interface unit for configuring the programmable logic.

31. The configurable processor system unit of claim 1, wherein the processor includes memory-mapped internal registers that are exported to the internal system bus.

32. The configurable processor system unit of claim 31, further comprising a signal interface for translating addresses permitting the internal registers to be exported.

33. The configurable processor system unit of claim 32, wherein the signal interface further includes an export table including a

40. The configurable processor system unit of claim 36, wherein the at least one region of programmable logic comprises short lines that connect adjacent tiles within a bank.

41. The configurable processor system unit of claim 40, wherein the at least one region of programmable logic further comprises long lines extending across multiple tiles in a bank.

42. The configurable processor system unit of claim 41, wherein the breaker tiles comprise programmable connections between the long lines between adjacent banks.

43. The configurable processor system unit of claim 36, wherein the bus is distributed such that a copy of the bus is distributed to each bank across the breaker tiles.

44. The configurable processor system unit of claim 43, wherein:

an address bus is distributed down the column of the horizontal breakers;

a write data bus is distributed down the column of the vertical breakers;

a read data bus is distributed up the column of vertical breakers.

45. The configurable processor system unit of claim 44, wherein additional global signals are distributed across the programmable logic using the breaker tiles.

46. The configurable processor system unit of claim 35, wherein a bank comprises an 8 tile by 8 tile row by column array of logic block tiles.

53. The configurable processor system unit of claim 52, wherein a single active signal from a single tile injected into the data path chain, such that one tile drives the signal.

54. The configurable processor system unit of claim 36, wherein a multi-source distributed signal is driven to the region of programmable logic by an array structure that allows a signal source to drive the array structure at any point, and provides a logical OR function of all sources to the array at any point of the array.

55. The configurable processor system unit of claim 54, wherein the multi-source distributed signal is a wait control signal.

56. The configurable processor system unit of claim 35, wherein the region of programmable logic further includes a configuration memory space used to configure the region of programmable logic.

57. The configurable processor system unit of claim 56, wherein the processor can address the configuration memory space as part of the system address space.

58. The configurable processor system unit of claim 56, wherein a bank of the region of programmable logic may be disconnected from other banks and may be reconfigured.

59. The configurable processor system unit of claim 1, wherein the bus is a multi-master shared bus, and the processor includes a hold signal to interact with the multi-master shared bus.

68. The configurable processor system unit of claim 67, wherein the master write segment comprises multiplexors for collecting individual master signals into the consolidated bus.

69. The configurable processor system unit of claim 64, wherein a slave write segment couples signals from the bus pipe to the programmable logic, and a slave read segment couples the signals from the programmable logic to the bus pipe.

70. The configurable processor system unit of claim 69, wherein the slave write segment couples the signals to selectors and pipe registers distributed throughout the programmable logic.

71. The configurable processor system unit of claim 70, wherein a socket interface write segment carries signals from the selectors and the pipe registers to the slaves within the programmable logic.

72. The configurable processor system unit of claim 71, wherein a socket interface read segment collects signals from the slaves to return pipe registers.

73. The configurable processor system unit of claim 72, wherein the signals collected from the slaves are routed through a general-purpose configurable logic of the programmable logic.

74. The configurable processor system unit of claim 72, wherein the socket interface read segment further comprises logical OR gates for collecting the signals into consolidated bus signals at the return pipe registers.

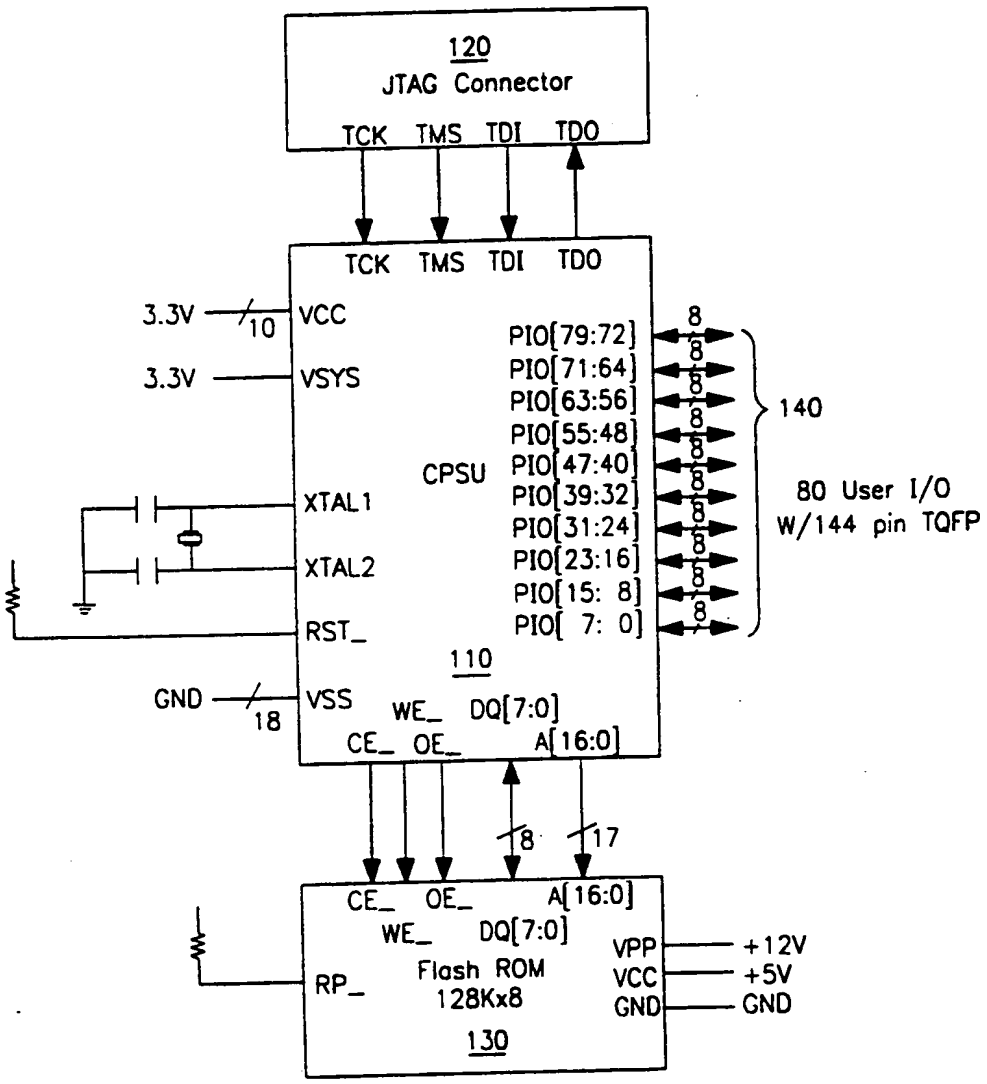

120
JTAG Connector
TCK TMS TDI TDO
TCK TMS TDI TDO
3.3V
10
VCC
3.3V
VSYS
CPSU
XTAL1
XTAL2
RST_
GND
18
VSS
110
PIO[79:72]
PIO[71:64]
PIO[63:56]
PIO[55:48]
PIO[47:40]
PIO[39:32]
PIO[31:24]
PIO[23:16]
PIO[15: 8]
PIO[7: 0]
8
140
80 User I/O
W/144 pin TQFP
WE_ DQ[7:0]
CE_ OE_ A[16:0]
8
17
CE_ OE_ A[16:0]
WE_ DQ[7:0]
RP_
Flash ROM
128Kx8
130
VPP
VCC
GND
+12V
+5V
GND

FIG. 1

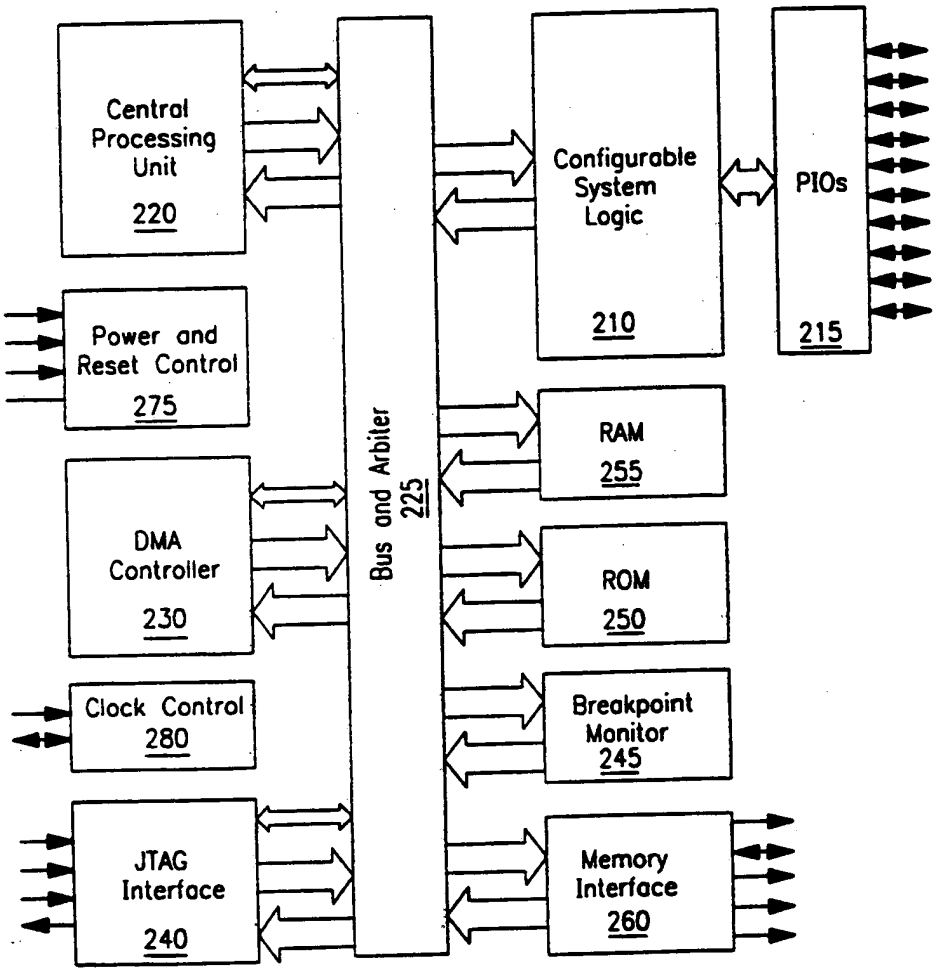

Central Processing Unit 220
Power and Reset Control 275
DMA Controller 230
Clock Control 280
JTAG Interface 240
Bus and Arbiter 225
Configurable System Logic 210
PIOs 215
RAM 255
ROM 250
Breakpoint Monitor 245
Memory Interface 260

FIG. 2

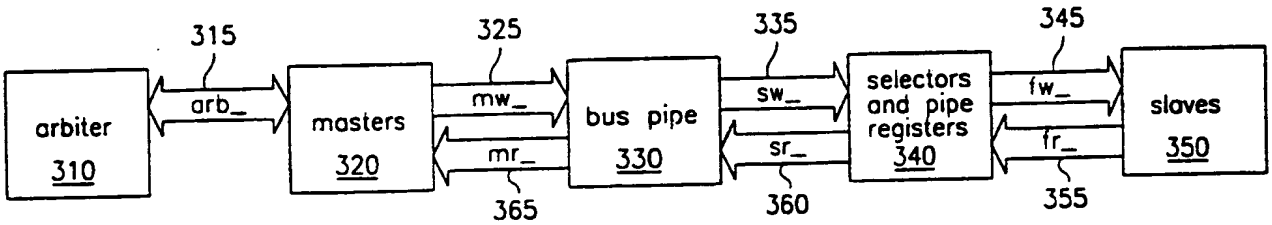

315
325
335
345
arbiter
310
arb_
masters
320
mw_
mr_
bus pipe
330
sw_
sr_
selectors
and pipe
registers
340
fw_
fr_
slaves
350
365
360
355

FIG. 3

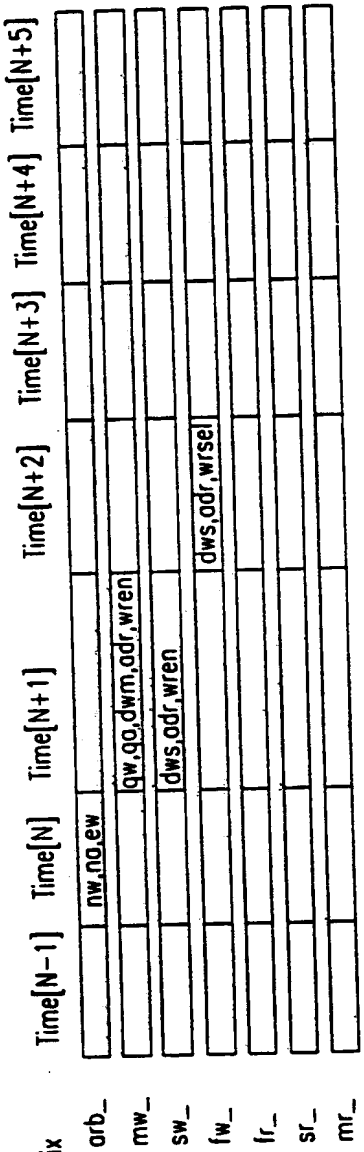

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
arb_
mw_
sw_
fw_
fr_
sr_
mr_
nw,no,ew
qw,qo,dwm,adr,wren
dws,adr,wren
dws,adr,wrsel

FIG. 4

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
arb_
mw_
sw_
fw_
fr_
sr_
mr_
nw,no,ew
qw,qo,dwm,adr,wren
dws,adr,wren
waitnow
dws,adr,wrsel
dws,adr,wrsel,waited
waitnow
waited

FIG. 5

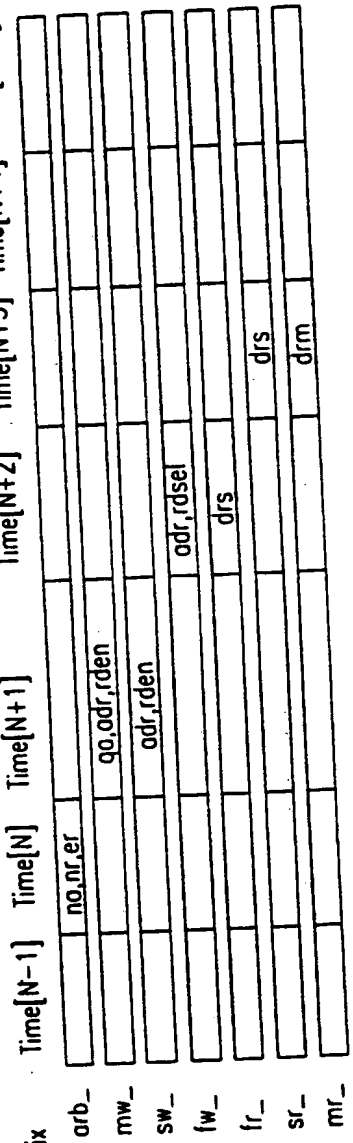

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
orb_
mw_
sw_
fw_
fr_
sr_
mr_
no,nr,er
go,adr,rden
adr,rden
adr,rdsel
drs
drs
drm

FIG. 6

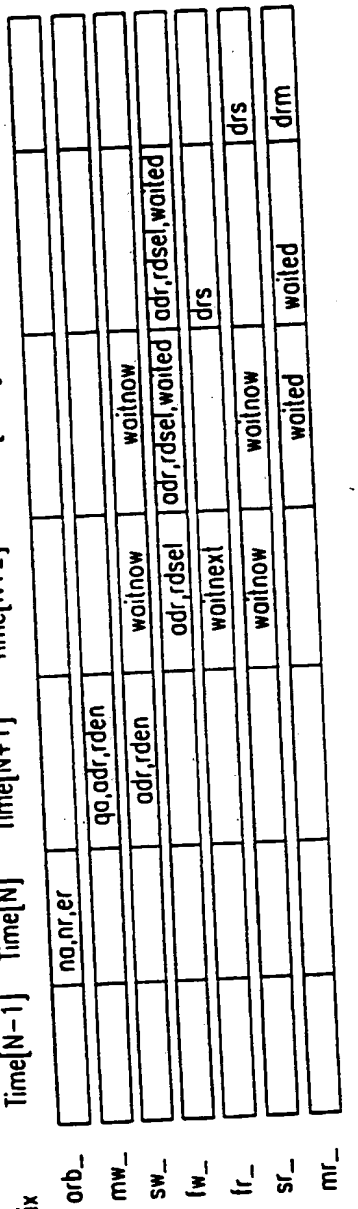

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
orb_
mw_
sw_
fw_
fr_
sr_
mr_
no,nr,er
go,adr,rden
adr,rden
waitnow
adr,rdsel
adr,rdsel,waited
waitnow
adr,rdsel,waited
waitnext
drs
waitnow
waitnow
drs
waited
waited
drm

FIG. 7

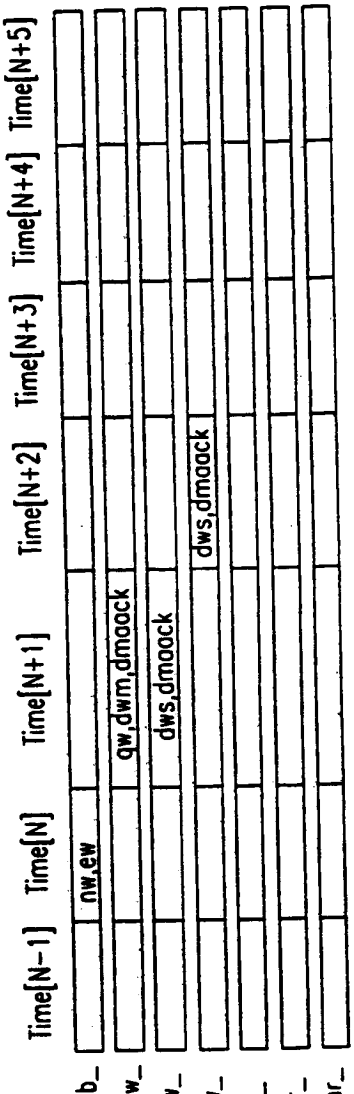

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
arb_
mw_
sw_
fw_
fr_
sr_
mr_
nw,ew
qw,dwm,dmaack
dws,dmaack
dws,dmaack

FIG. 8

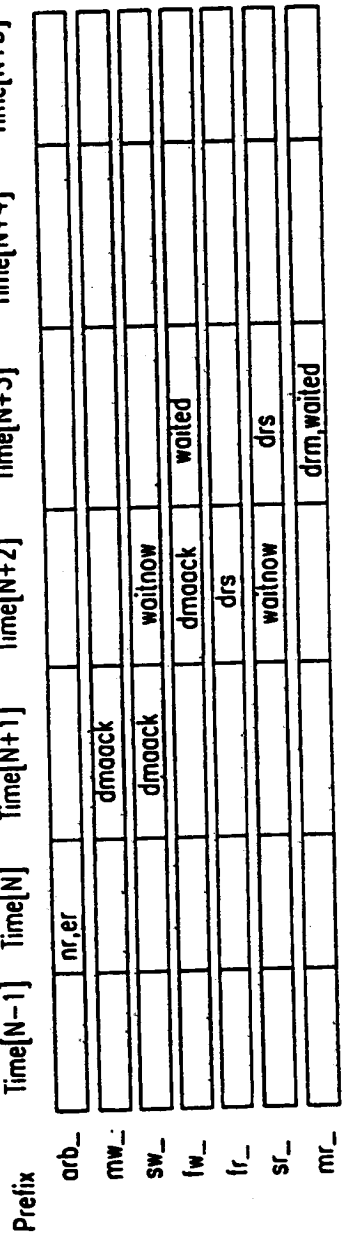

Prefix
Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
arb_
mw_
sw_
fw_
fr_
sr_
mr_
nr,er
dmaack
dmaack
dmaack
drs
waitnow
waitnow
waited
drs
drm,waited

FIG. 9

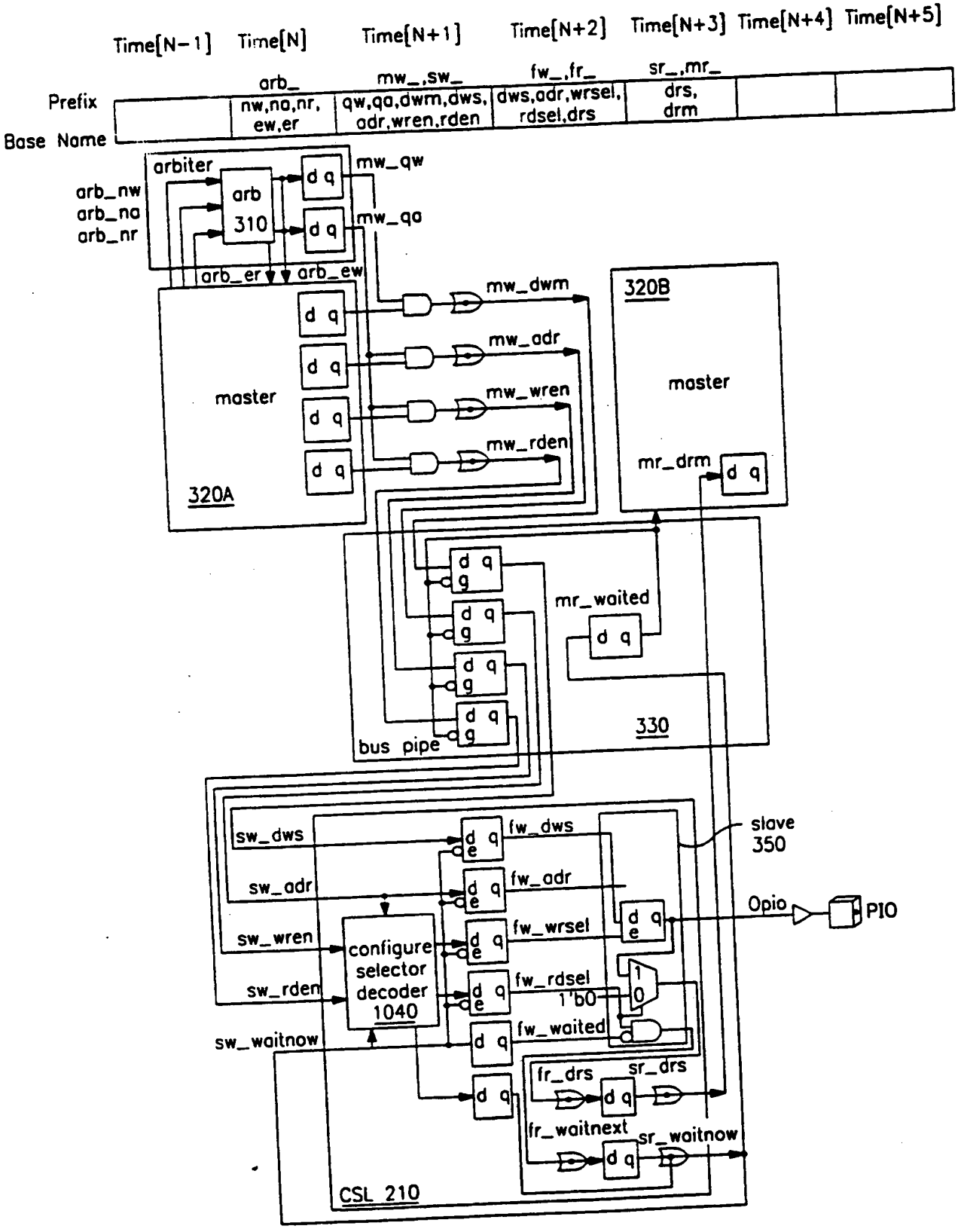

Time[N-1]
Time[N]
Time[N+1]
Time[N+2]
Time[N+3]
Time[N+4]
Time[N+5]
Prefix
Base Name
arb_ nw,na,nr, ew,er
mw_,sw_ qw,qa,dwm,dws, adr,wren,rden
fw_,fr_ dws,adr,wrsel, rdsel,drs
sr_,mr_ drs, drm
arbiter
arb 310
arb_nw
arb_na
arb_nr
arb_er
arb_ew
mw_qw
mw_qa
mw_dwm
mw_adr
mw_wren
mw_rden
master
320A
320B
mr_drm
mr_waited
bus pipe
330
sw_dws
sw_adr
sw_wren
sw_rden
sw_waitnow
configure selector decoder 1040
fw_dws
fw_adr
fw_wrsel
fw_rdsel
1'b0
fw_waited
fr_drs
sr_drs
fr_waitnext
sr_waitnow
CSL 210
slave 350
Opio
PIO

FIG. 10

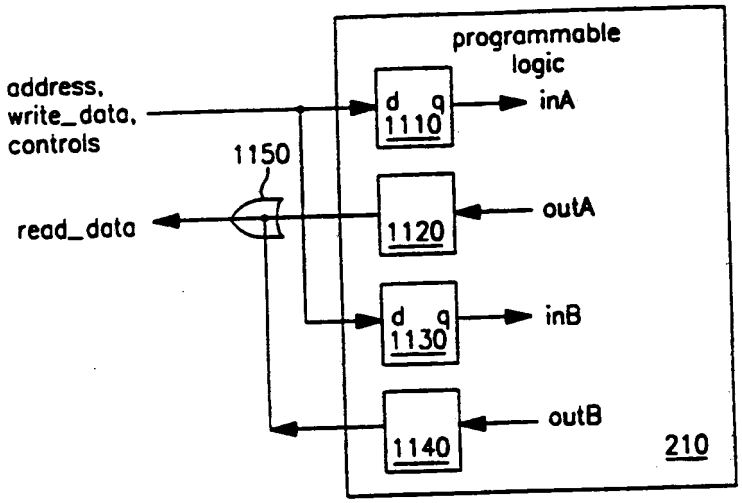

programmable
logic
address,
write_data,
controls
d q
1110
inA
1150
read_data
1120
outA
d q
1130
inB
1140
outB
210

FIG. 11

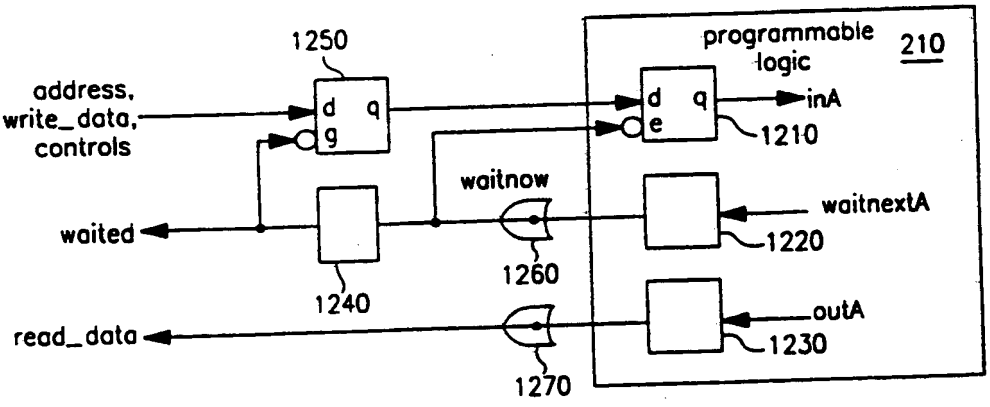

1250
programmable
logic
210
address,
write_data,
controls
d q
g
d q
e
inA
1210
waitnow
waitnextA
waited
1220
1260
1240
outA
read_data
1230
1270

FIG. 12

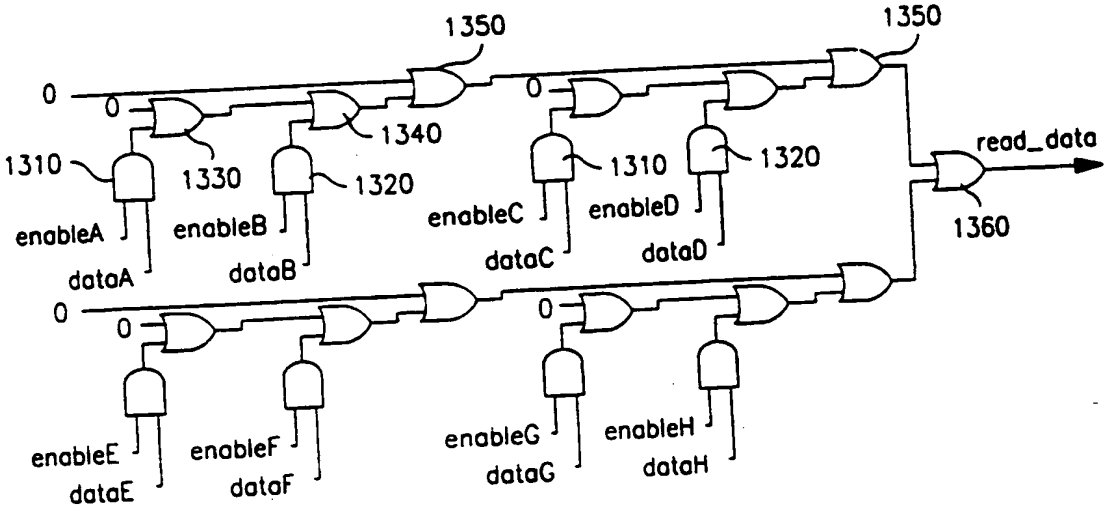

1350
1350
0
0
0
0
1340
1310
1330
1320
1310
1320
read_data
enableA
enableB
enableC
enableD
1360
dataA
dataB
dataC
dataD
0
0
0
0
enableE
enableF
enableG
enableH
dataE
dataF
dataG
dataH

FIG. 13

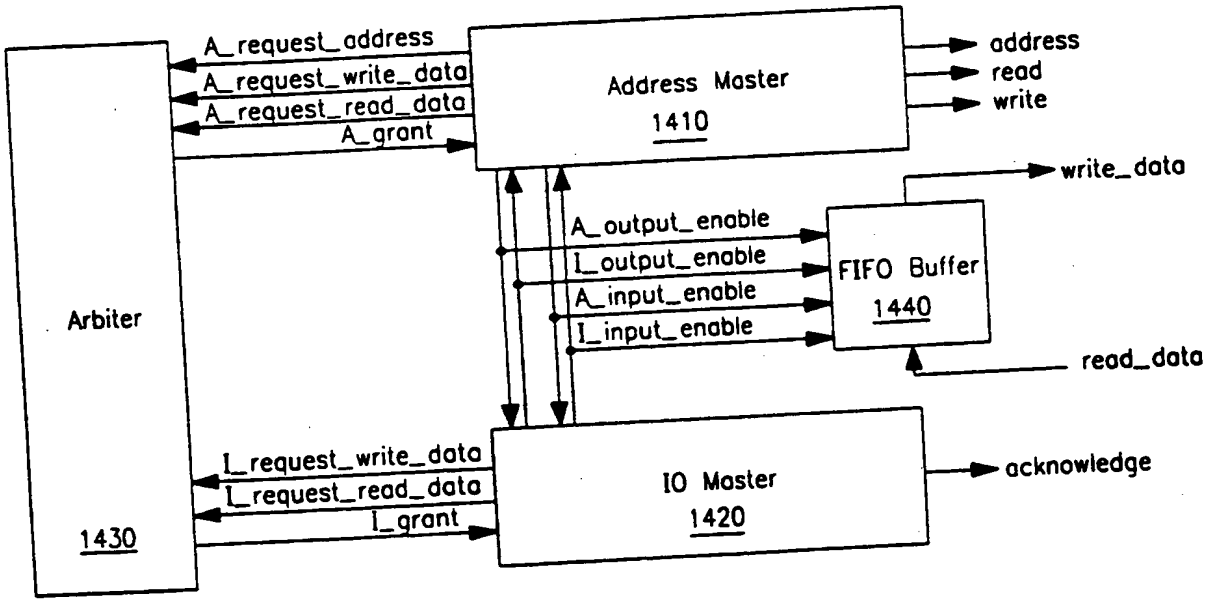

A_request_address
A_request_write_data
A_request_read_data
A_grant
Address Master
1410
address
read
write
write_data
A_output_enable
I_output_enable
A_input_enable
I_input_enable
FIFO Buffer
1440
read_data
Arbiter
1430
I_request_write_data
I_request_read_data
I_grant
IO Master
1420
acknowledge

FIG. 14

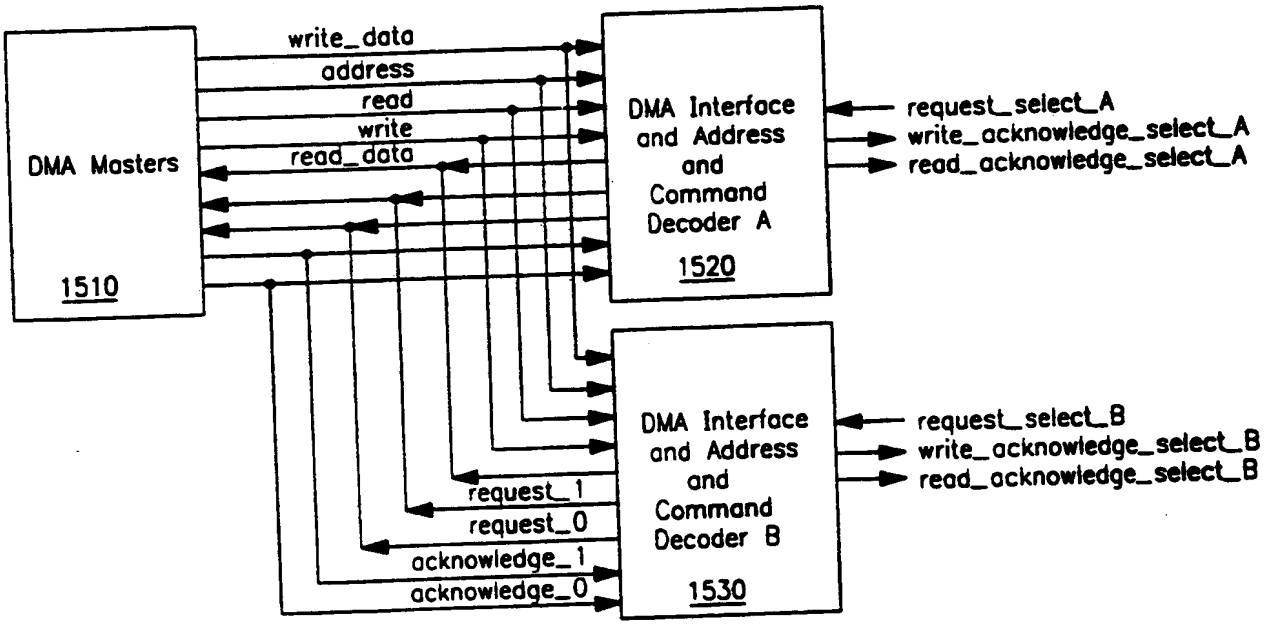

DMA Masters
1510
write_data
address
read
write
read_data
DMA Interface and Address and Command Decoder A
1520
request_select_A
write_acknowledge_select_A
read_acknowledge_select_A
DMA Interface and Address and Command Decoder B
1530
request_select_B
write_acknowledge_select_B
read_acknowledge_select_B
request_1
request_0
acknowledge_1
acknowledge_0

FIG. 15

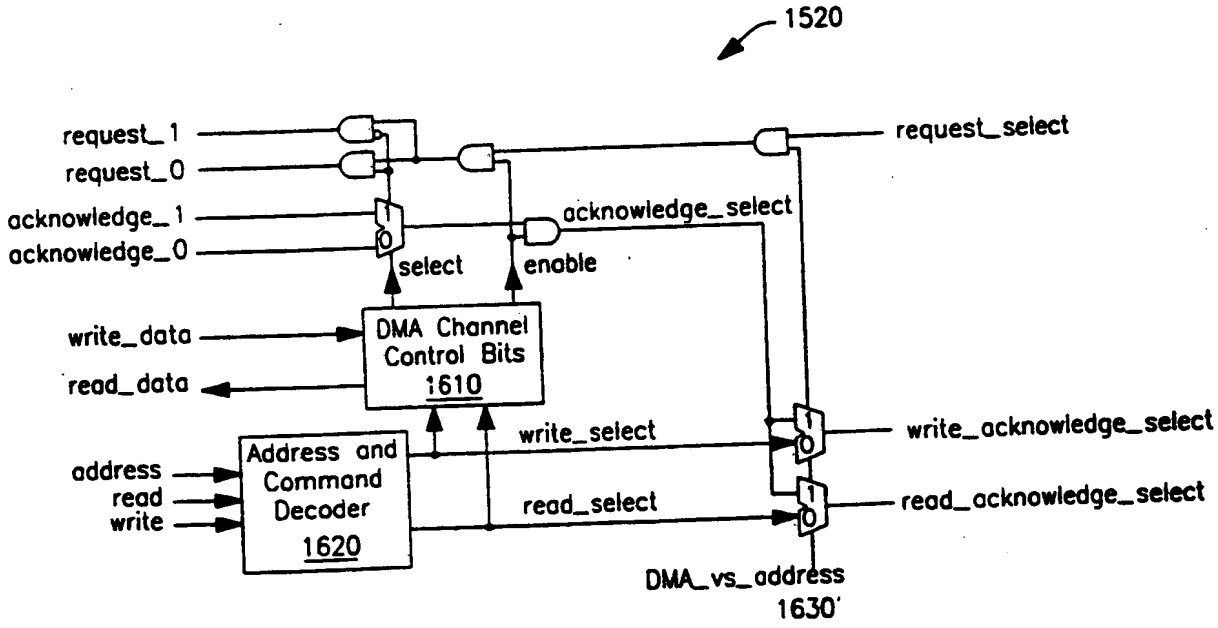

1520
request_1
request_0
acknowledge_1
acknowledge_0
select
enable
request_select
acknowledge_select
write_data
read_data
DMA Channel Control Bits
1610
address
read
write
Address and Command Decoder
1620
write_select
read_select
write_acknowledge_select
read_acknowledge_select
DMA_vs_address
1630

FIG. 16

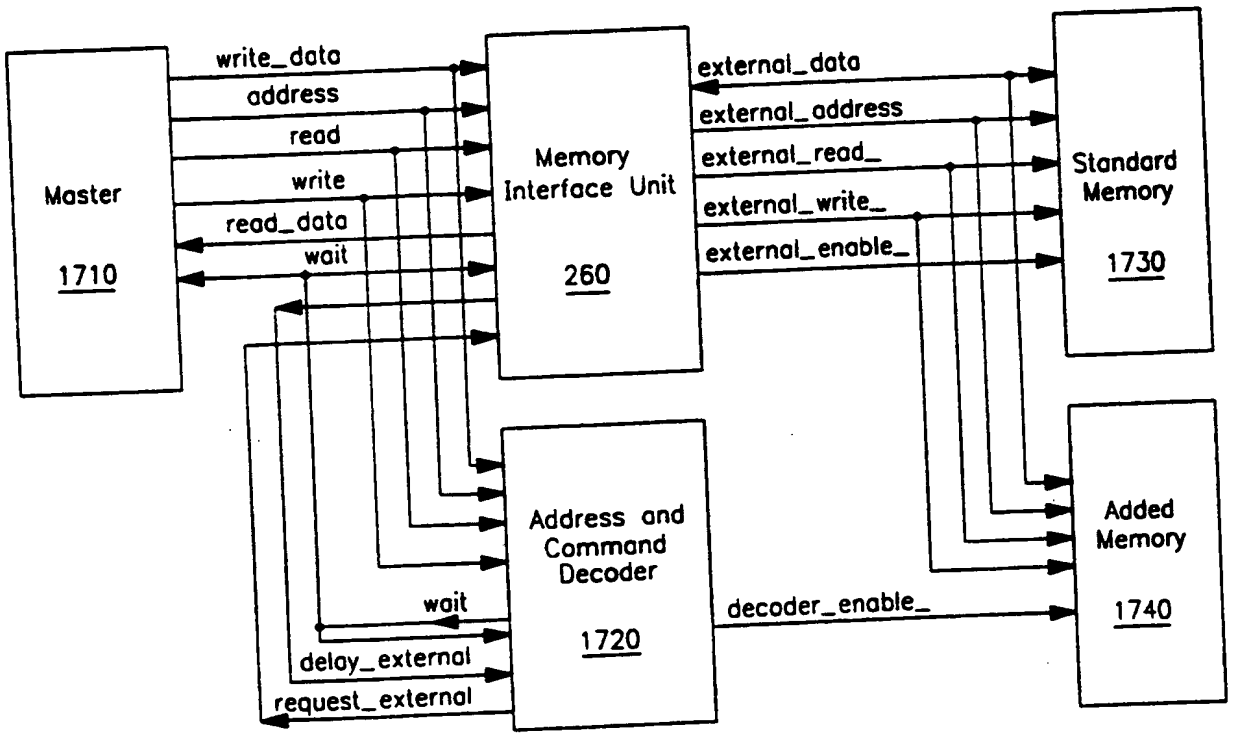

Master
1710
write_data
address
read
write
read_data
wait
Memory
Interface Unit
260
external_data
external_address
external_read_
external_write_
external_enable_
Standard
Memory
1730
Address and
Command
Decoder
1720
wait
delay_external
request_external
decoder_enable_
Added
Memory
1740

FIG. 17

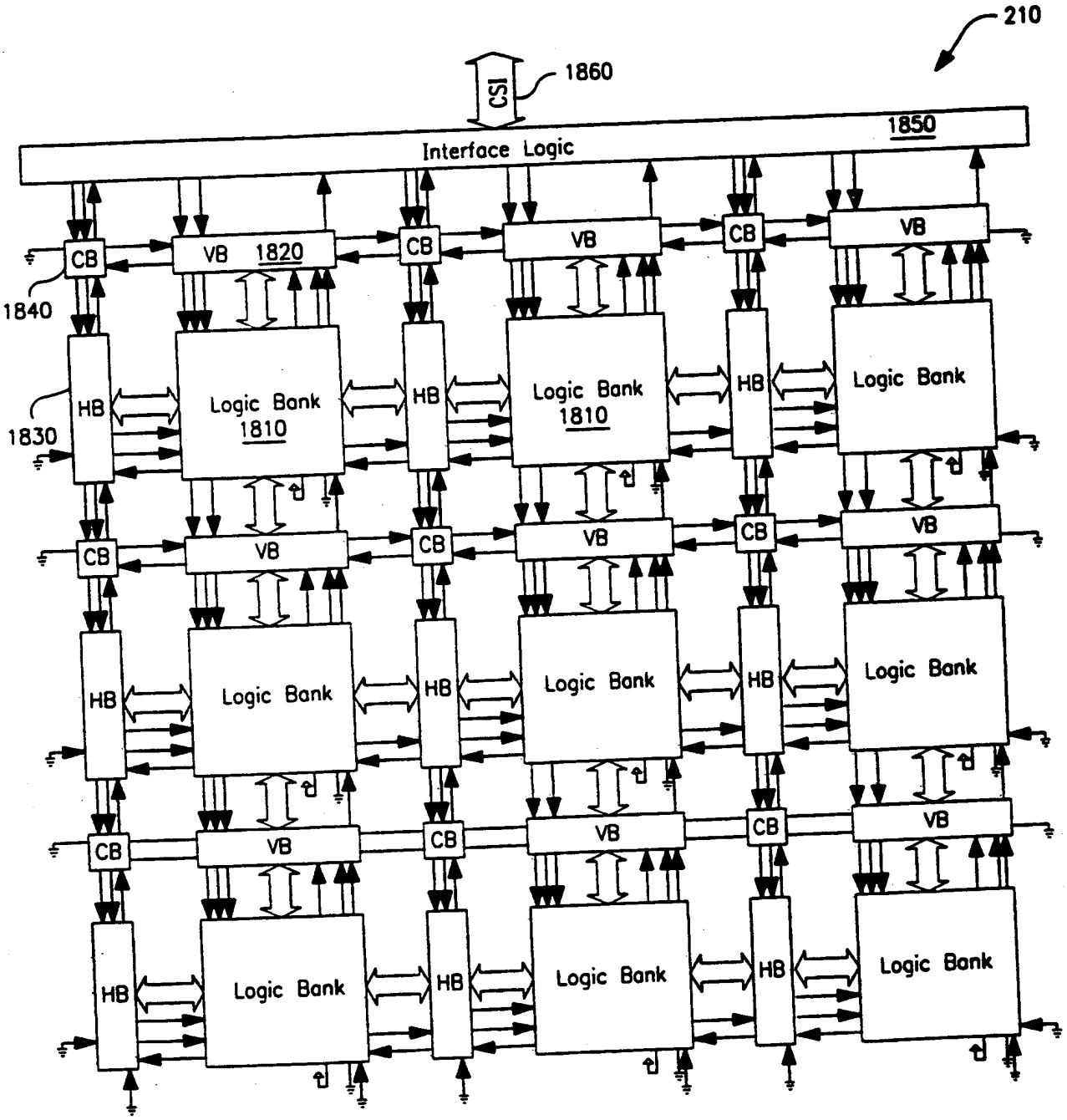

210
CSI
1860
Interface Logic
1850
CB
VB
1820
1840
HB
Logic Bank
1810
1830

FIG. 18

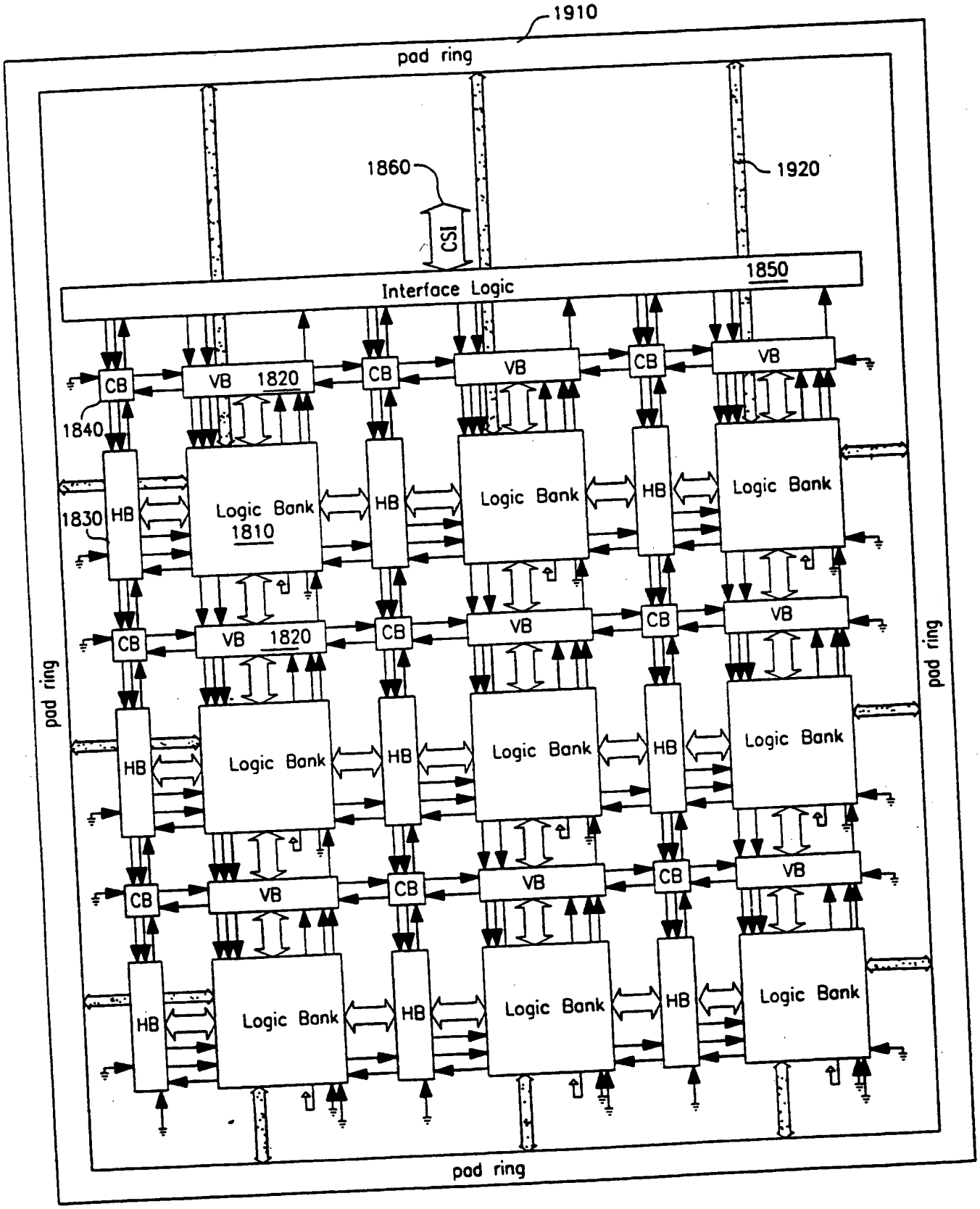

1910
pad ring
1860
CSI
1920
Interface Logic
1850
CB
VB 1820
VB
1840
1830
HB
Logic Bank
1810
pad ring
pad ring

FIG. 19

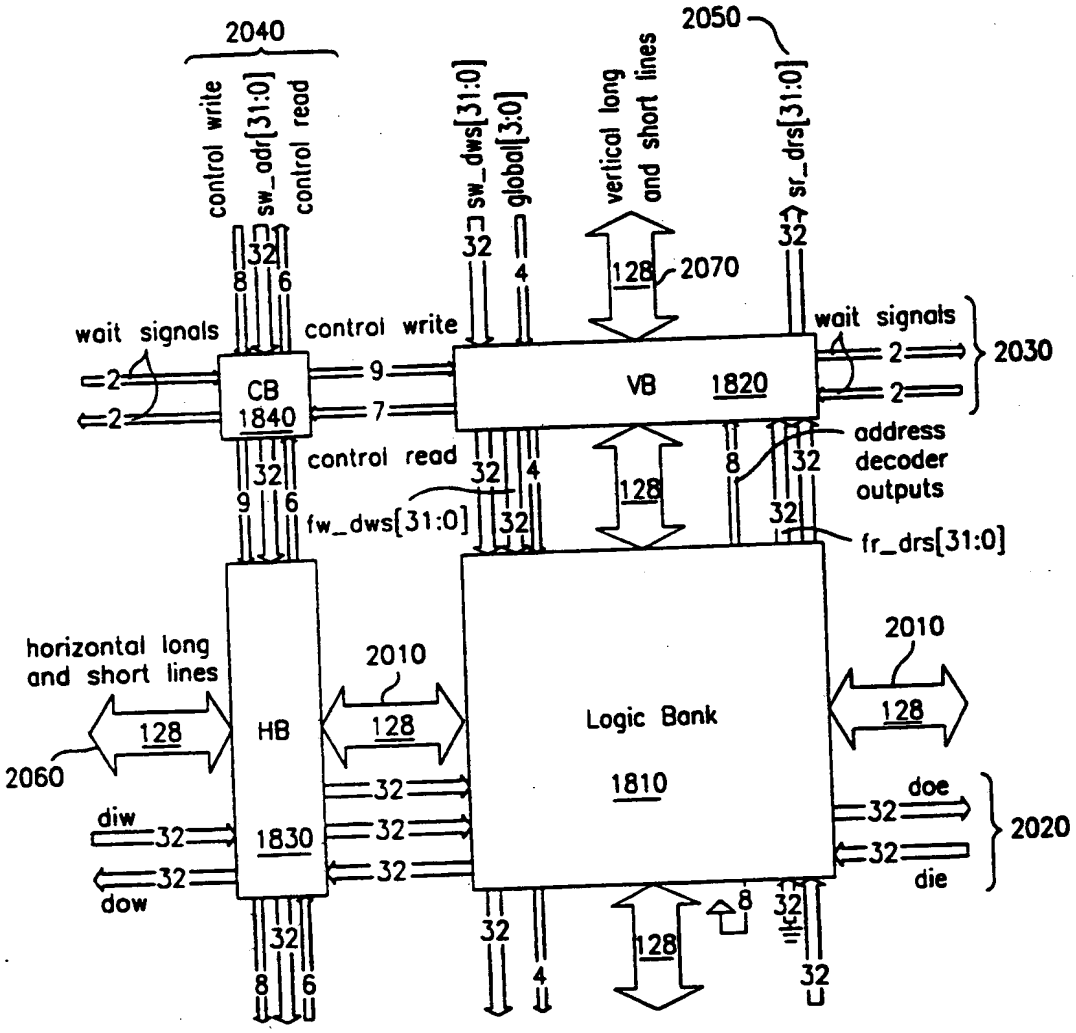

2040
control write
sw_adr[31:0]
control read
sw_dws[31:0]
global[3:0]
vertical long and short lines
2050
sr_drs[31:0]
32
8
6
4
128
2070
wait signals
control write
CB
1840
9
7
VB
1820
wait signals
2
2030
address decoder outputs
control read
fw_dws[31:0]
fr_drs[31:0]
horizontal long and short lines
2010
HB
1830
Logic Bank
1810
2060
diw
dow
doe
die
2020

FIG. 20

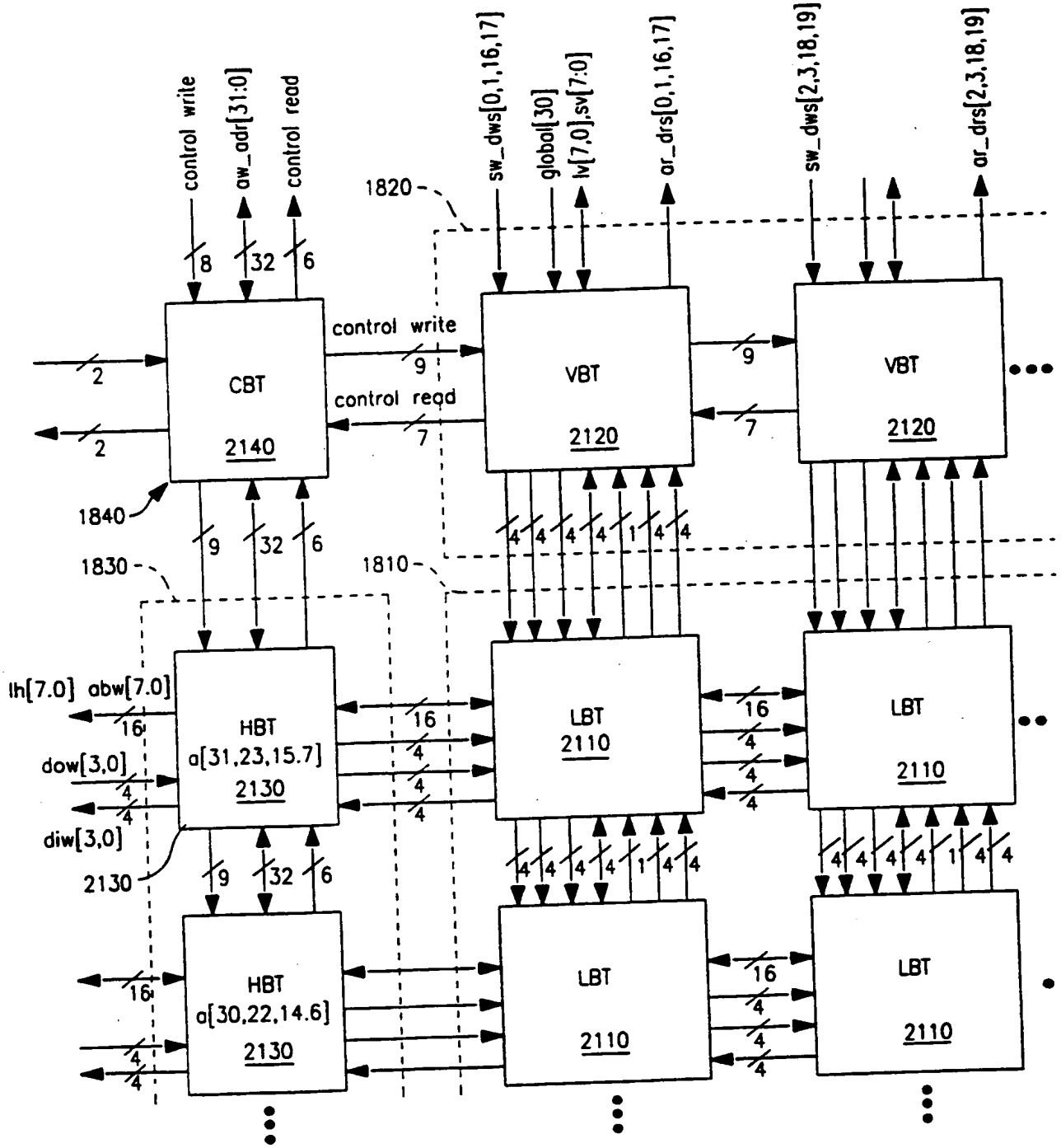

control write
aw_adr[31:0]
control read
sw_dws[0,1,16,17]
global[30]
lv[7:0],sv[7:0]
ar_drs[0,1,16,17]
sw_dws[2,3,18,19]
ar_drs[2,3,18,19]
1820
8
32
6
CBT
2140
control write
control read
9
7
VBT
2120
VBT
2120
2
2
1840
1830
1810
9
32
6
4 4 4 4 1 4 4
lh[7.0] abw[7.0]
16
HBT
a[31,23,15.7]
2130
dow[3,0]
4
diw[3,0]
2130
LBT
2110
LBT
2110
HBT
a[30,22,14.6]
2130
LBT
2110
LBT
2110

FIG. 21

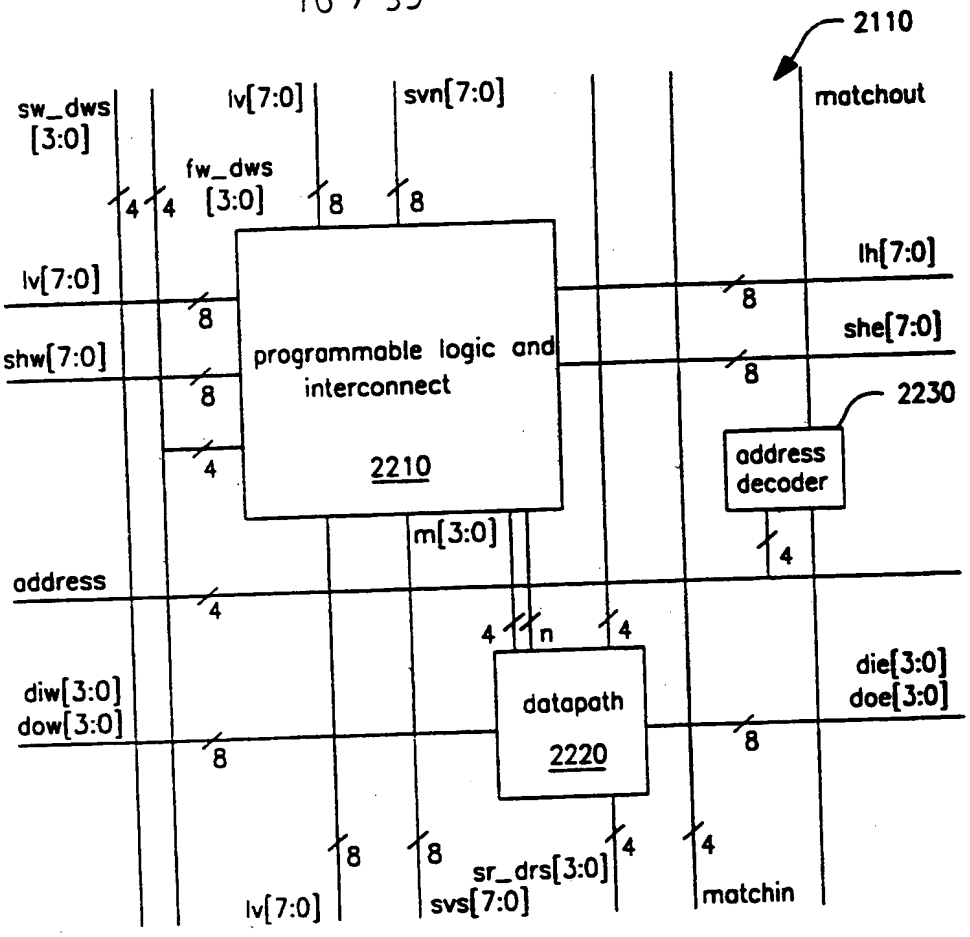


FIG. 22A

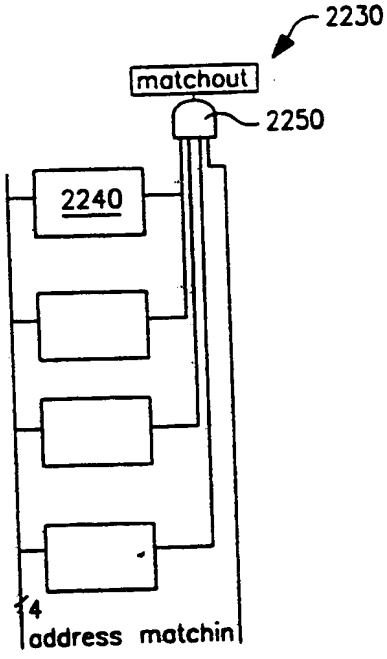


FIG. 22B

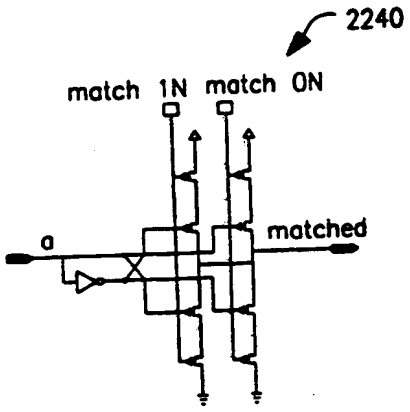


FIG. 22C

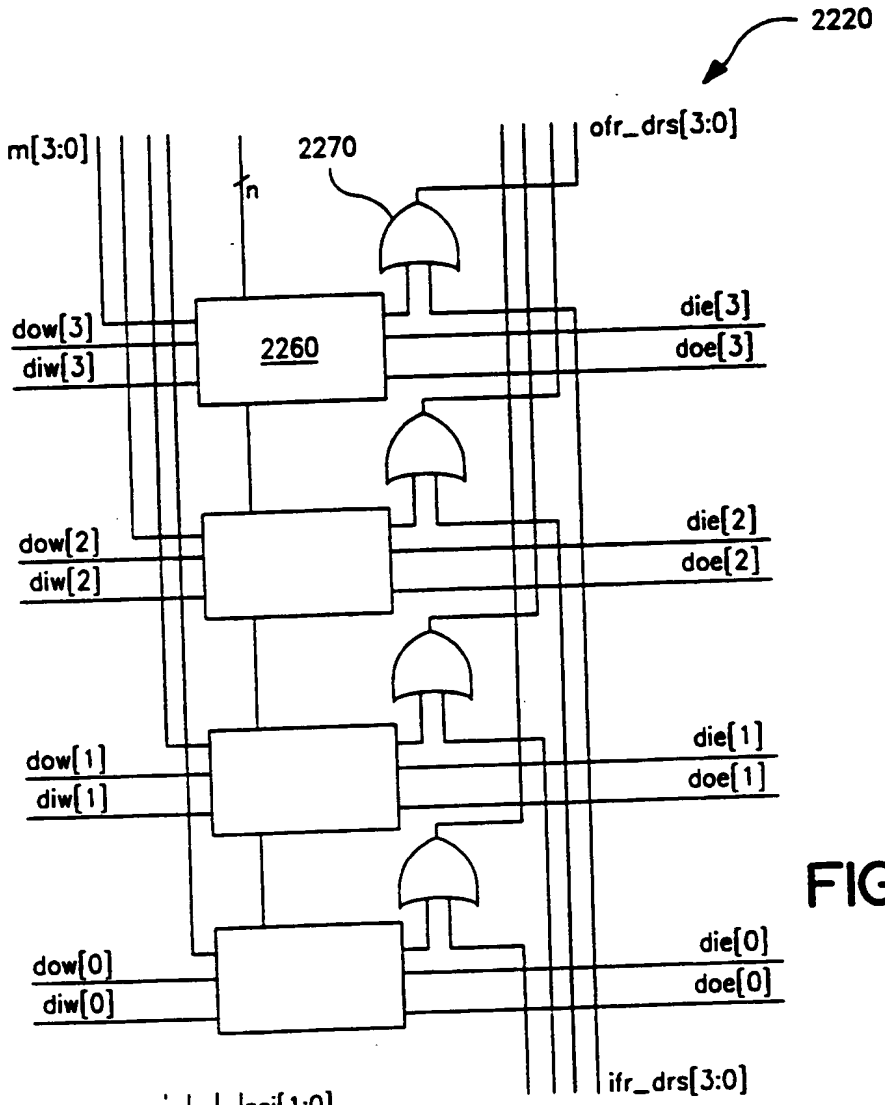


FIG. 22D

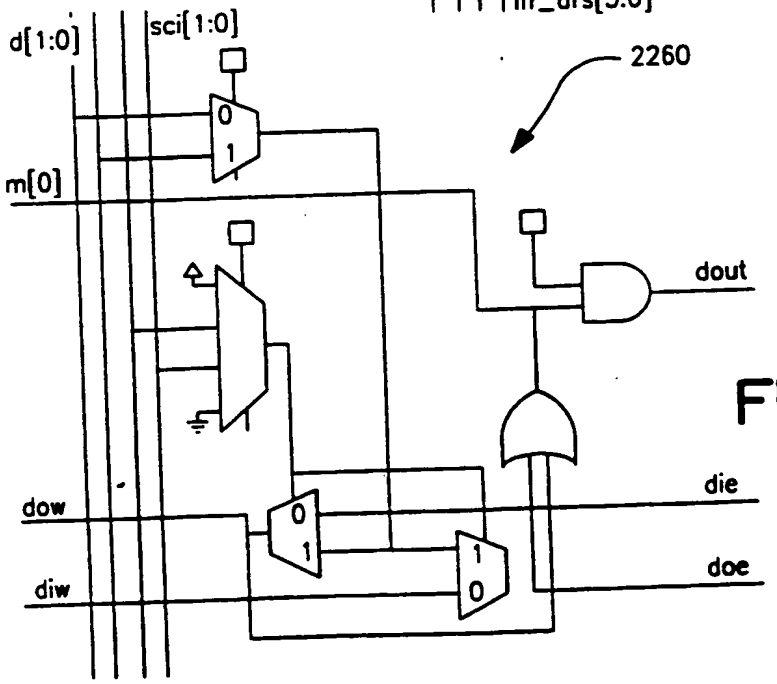


FIG. 22E

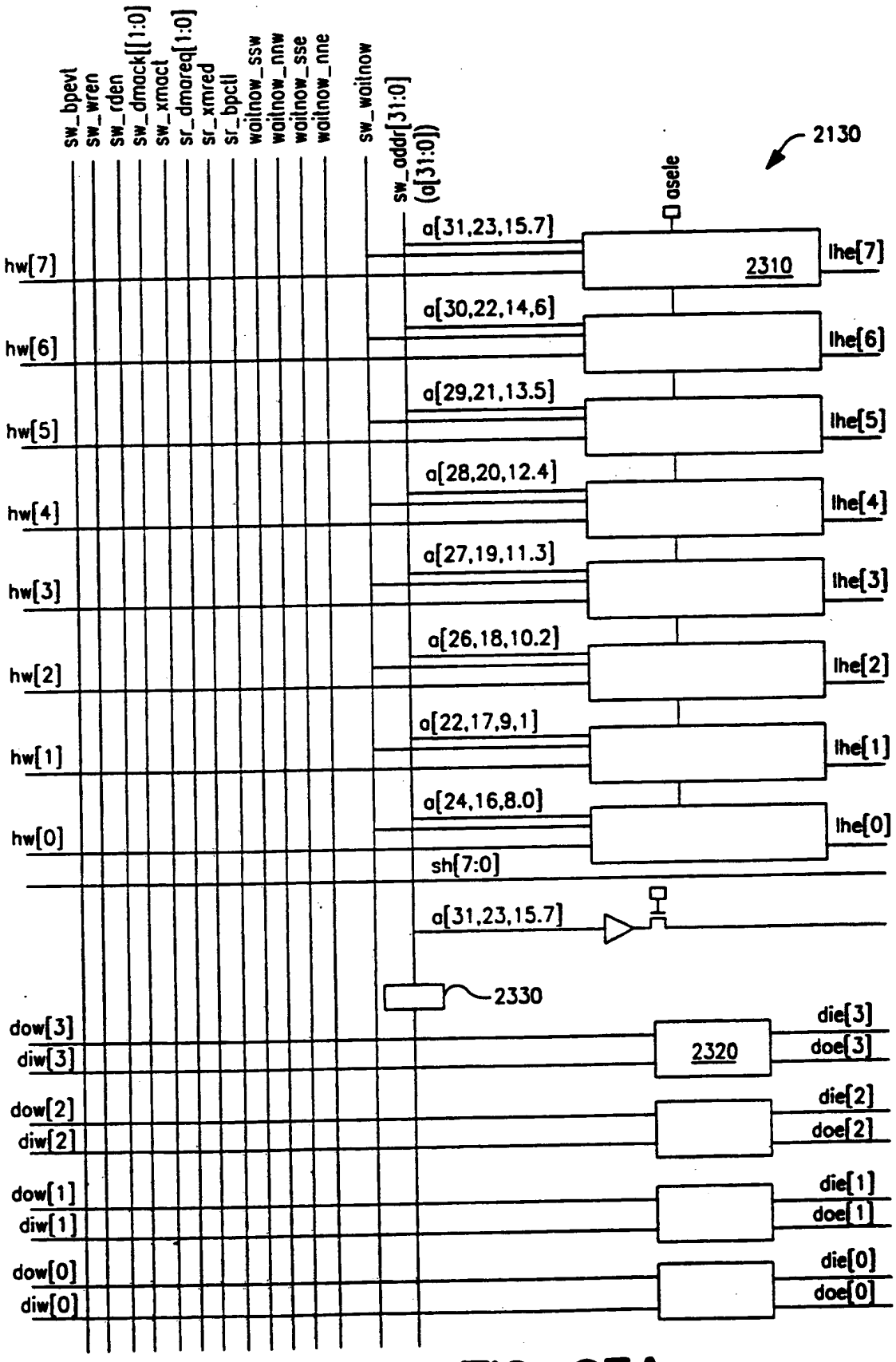

sw_bpevt
sw_wren
sw_rden
sw_dmack[1:0]
sw_xmact
sr_dmareq[1:0]
sr_xmred
sr_bpctl
wainow_ssw
wainow_nnw
wainow_sse
wainow_nne
sw_wainow
sw_addr[31:0]
(a[31:0])
2130
asele
a[31,23,15.7]
2310
lhe[7]
hw[7]
a[30,22,14,6]
lhe[6]
hw[6]
a[29,21,13.5]
lhe[5]
hw[5]
a[28,20,12.4]
lhe[4]
hw[4]
a[27,19,11.3]
lhe[3]
hw[3]
a[26,18,10.2]
lhe[2]
hw[2]
a[22,17,9,1]
lhe[1]
hw[1]
a[24,16,8.0]
lhe[0]
hw[0]
sh[7:0]
a[31,23,15.7]
2330
dow[3]
diw[3]
2320
die[3]
doe[3]
dow[2]
diw[2]
die[2]
doe[2]
dow[1]
diw[1]
die[1]
doe[1]
dow[0]
diw[0]
die[0]
doe[0]

FIG. 23A

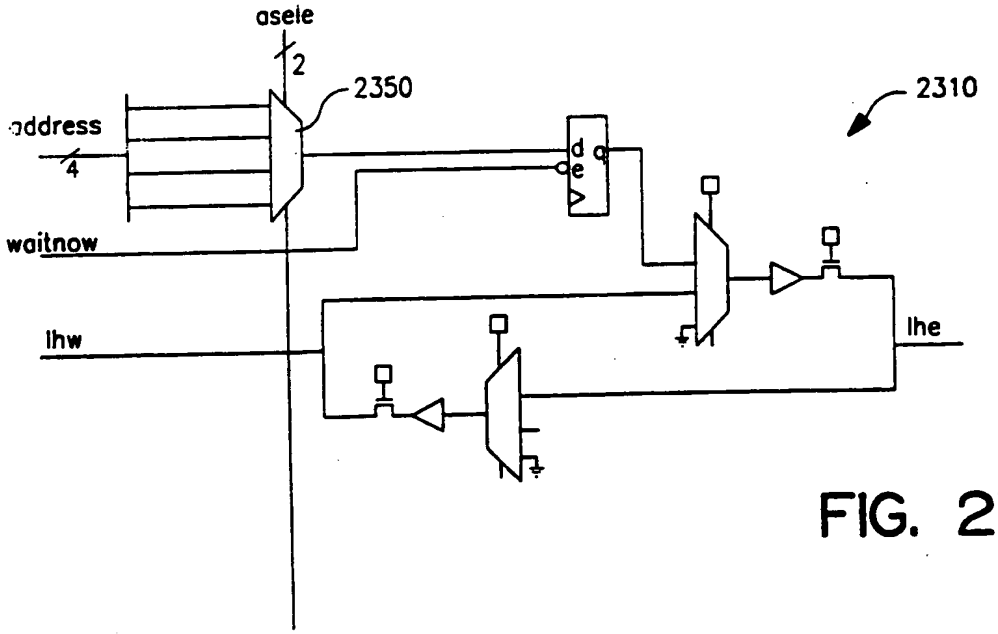


FIG. 23B

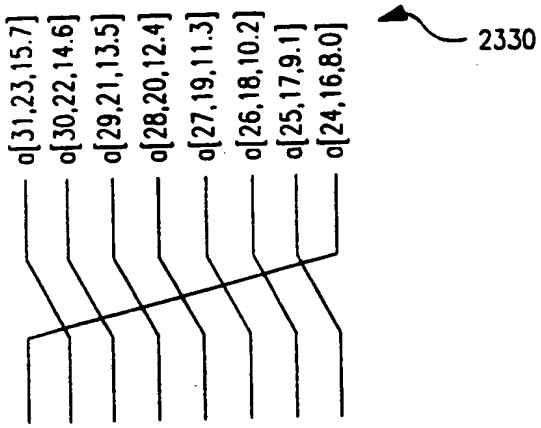


FIG. 23C

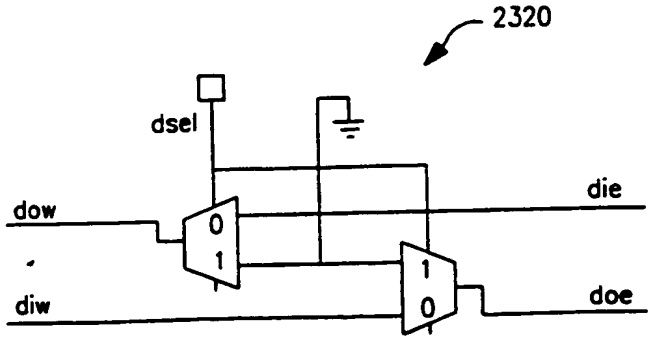


FIG. 23D

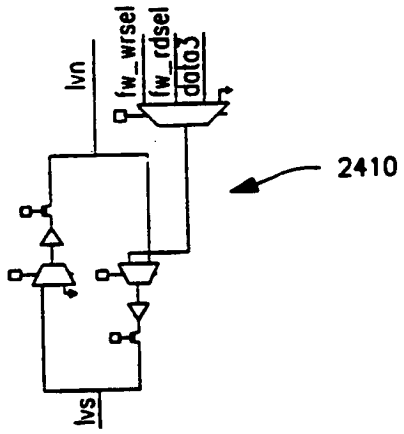


FIG. 24A

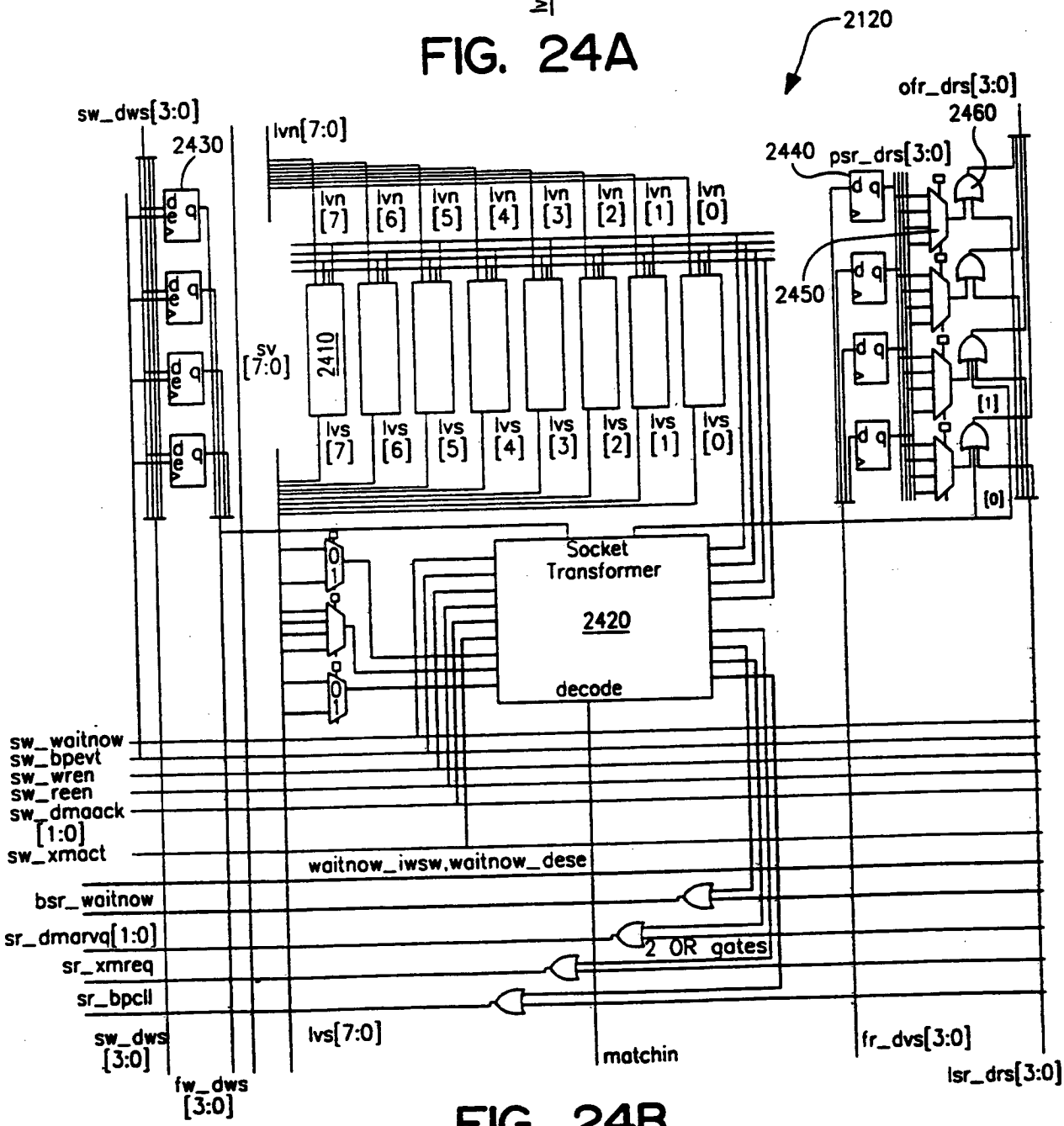


FIG. 24B

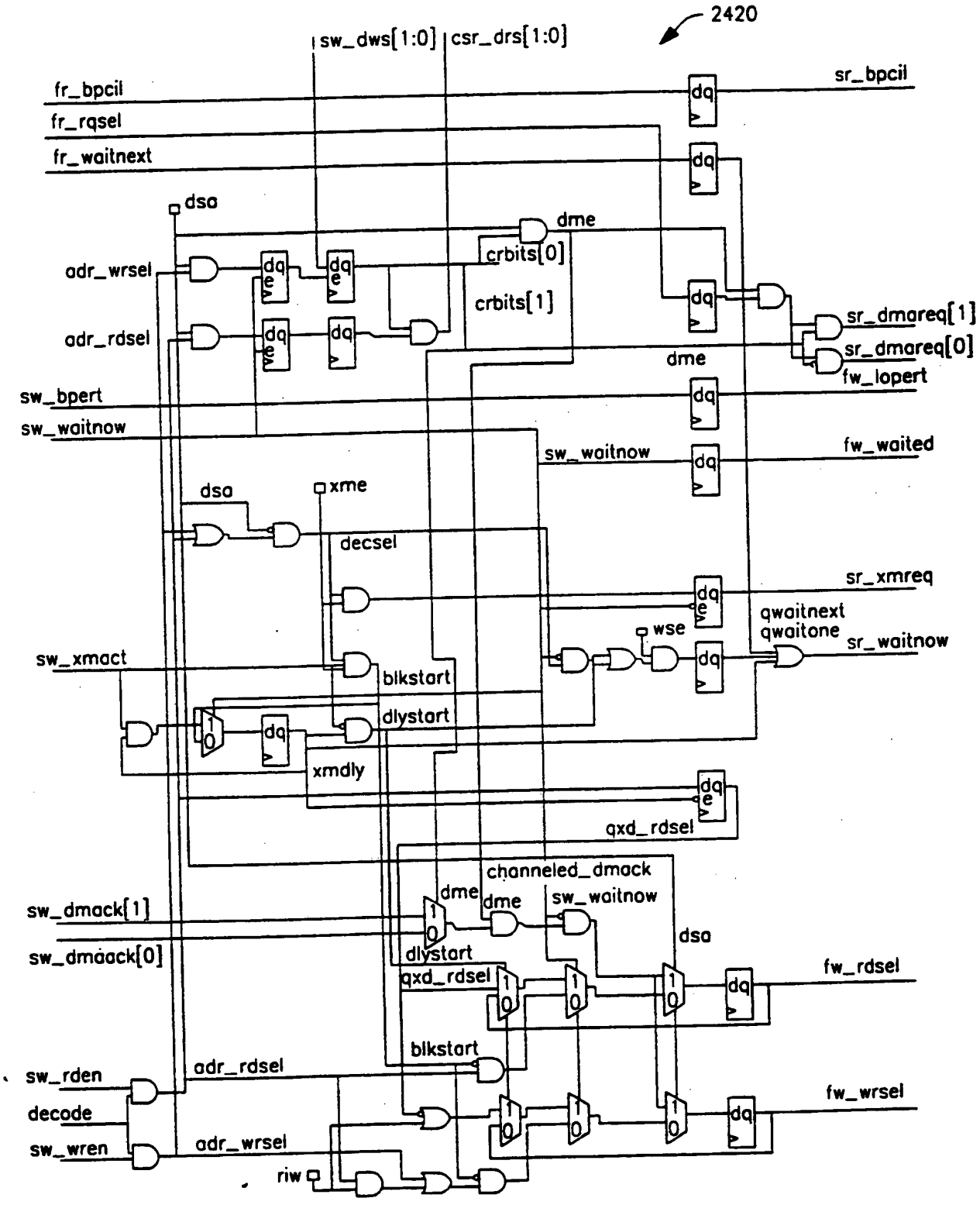

2420
sw_dws[1:0]
csr_drs[1:0]
fr_bpcil
fr_rqsel
fr_waitnext
sr_bpcil
dsa
dme
adr_wrsel
adr_rdsel
crbits[0]
crbits[1]
sr_dmareq[1]
sr_dmareq[0]
fw_lopert
sw_bpert
sw_waitnow
fw_waited
xme
decsel
sr_xmreq
qwaitnext
qwaitone
wse
sr_waitnow
sw_xmact
blkstart
dlystart
xmdly
qxd_rdsel
channeled_dmack
sw_dmack[1]
sw_dmaack[0]
fw_rdsel
sw_rden
decode
sw_wren
adr_rdsel
adr_wrsel
fw_wrsel
riw

FIG. 25

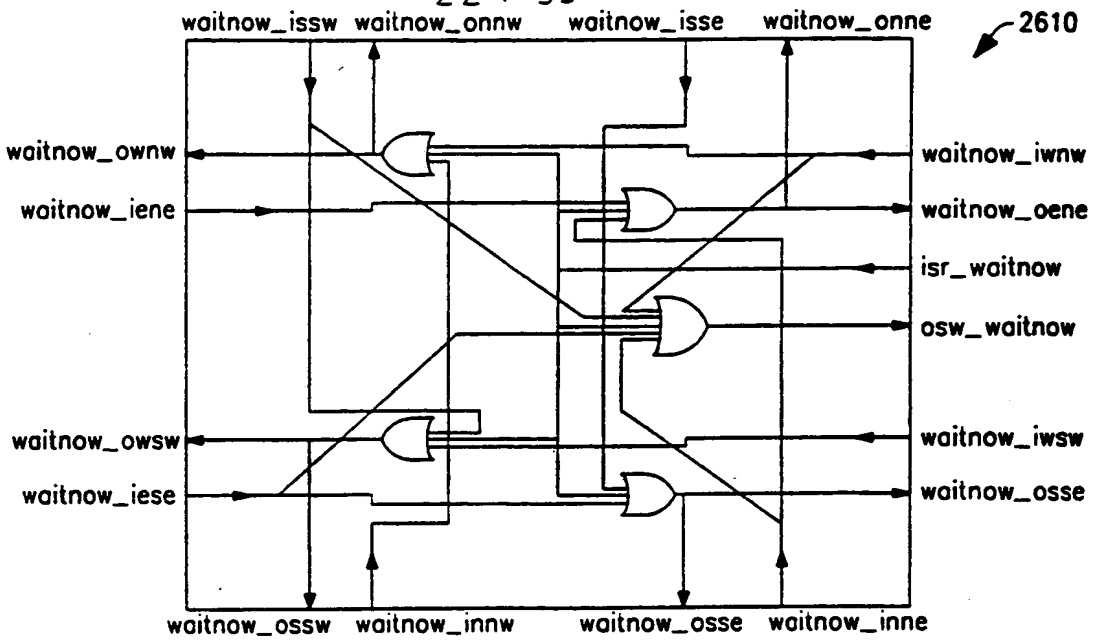


FIG. 26A

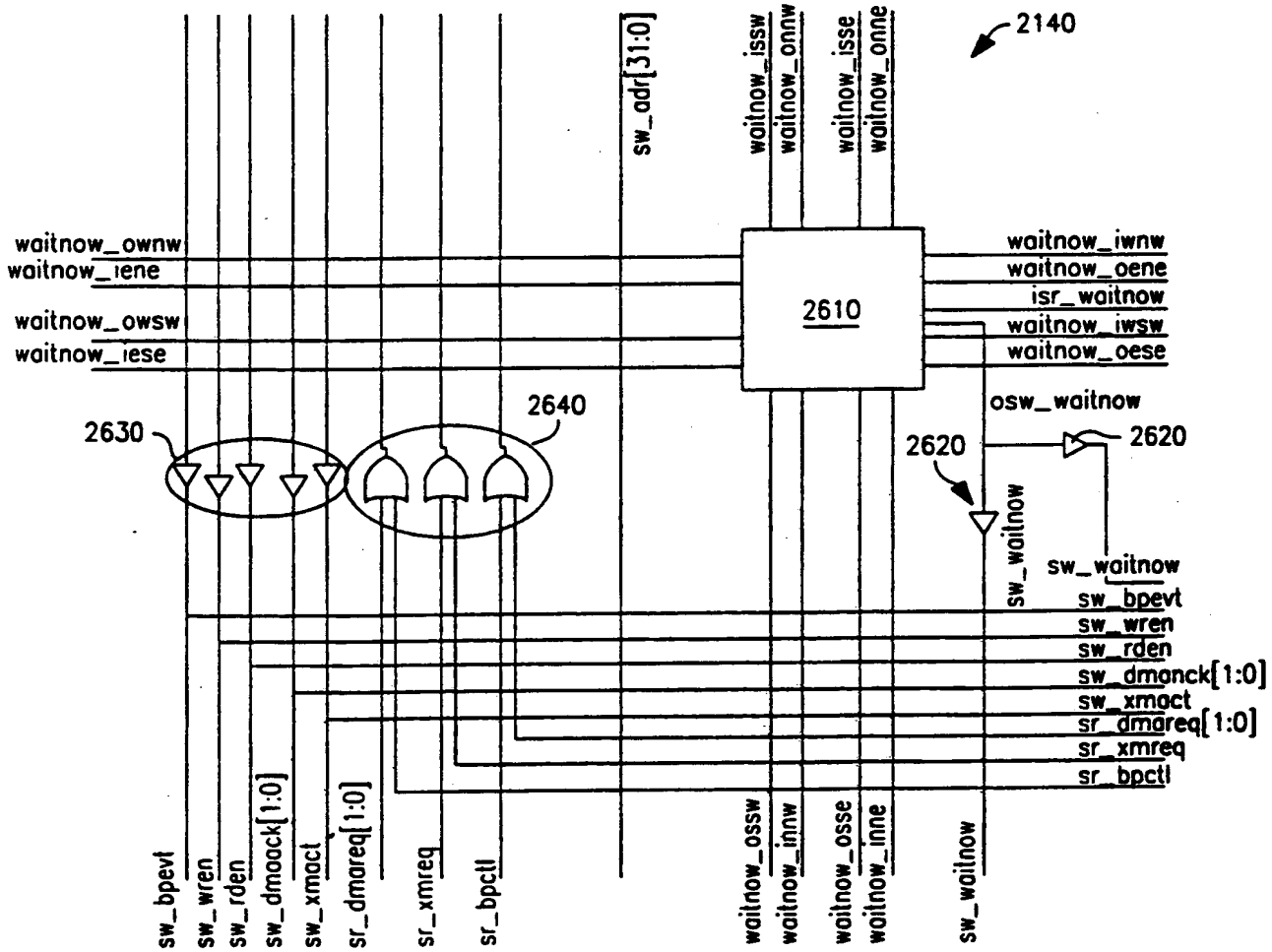


FIG. 26B

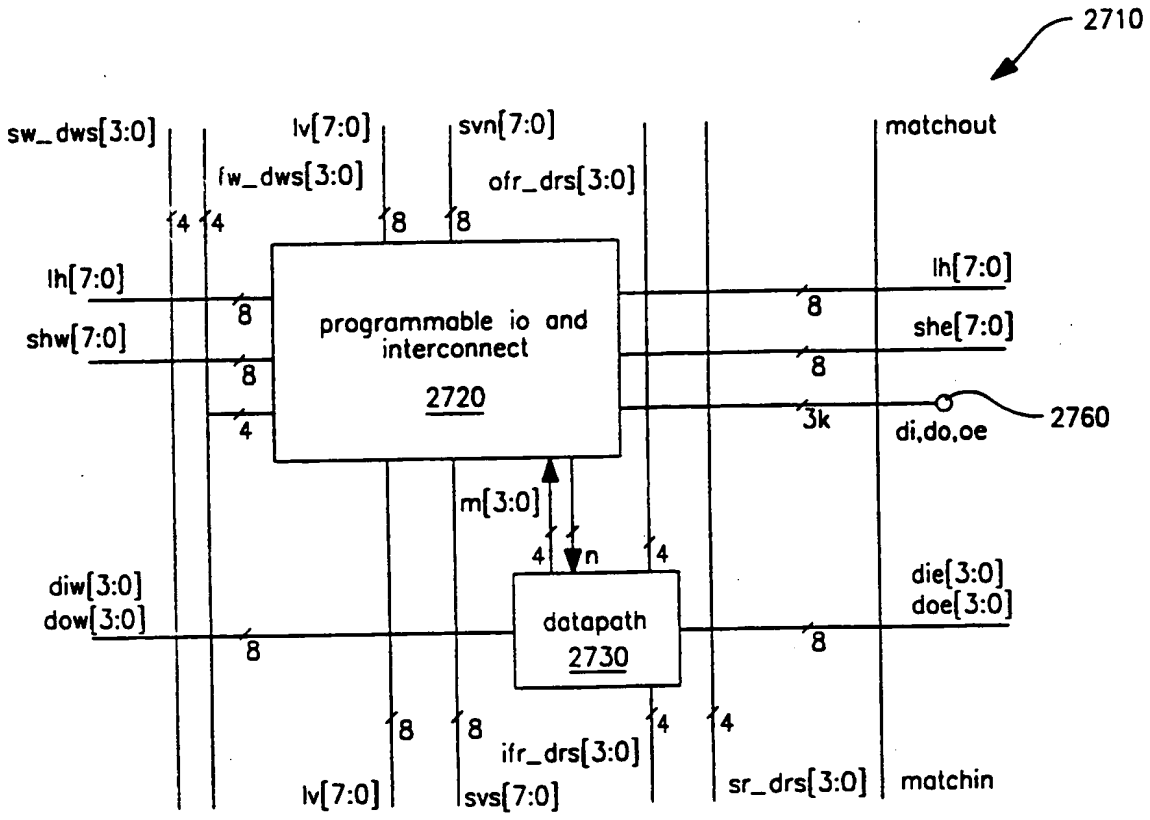


FIG. 27A

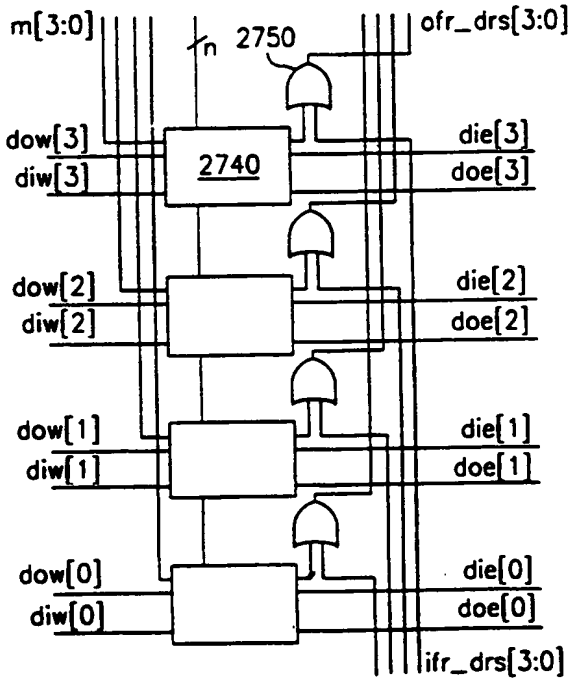


FIG. 27B

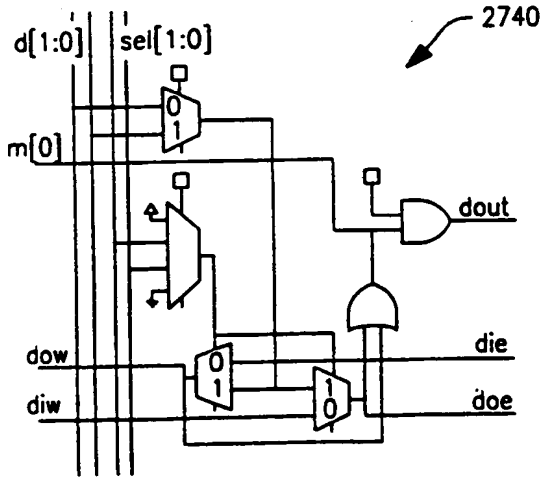


FIG. 27C

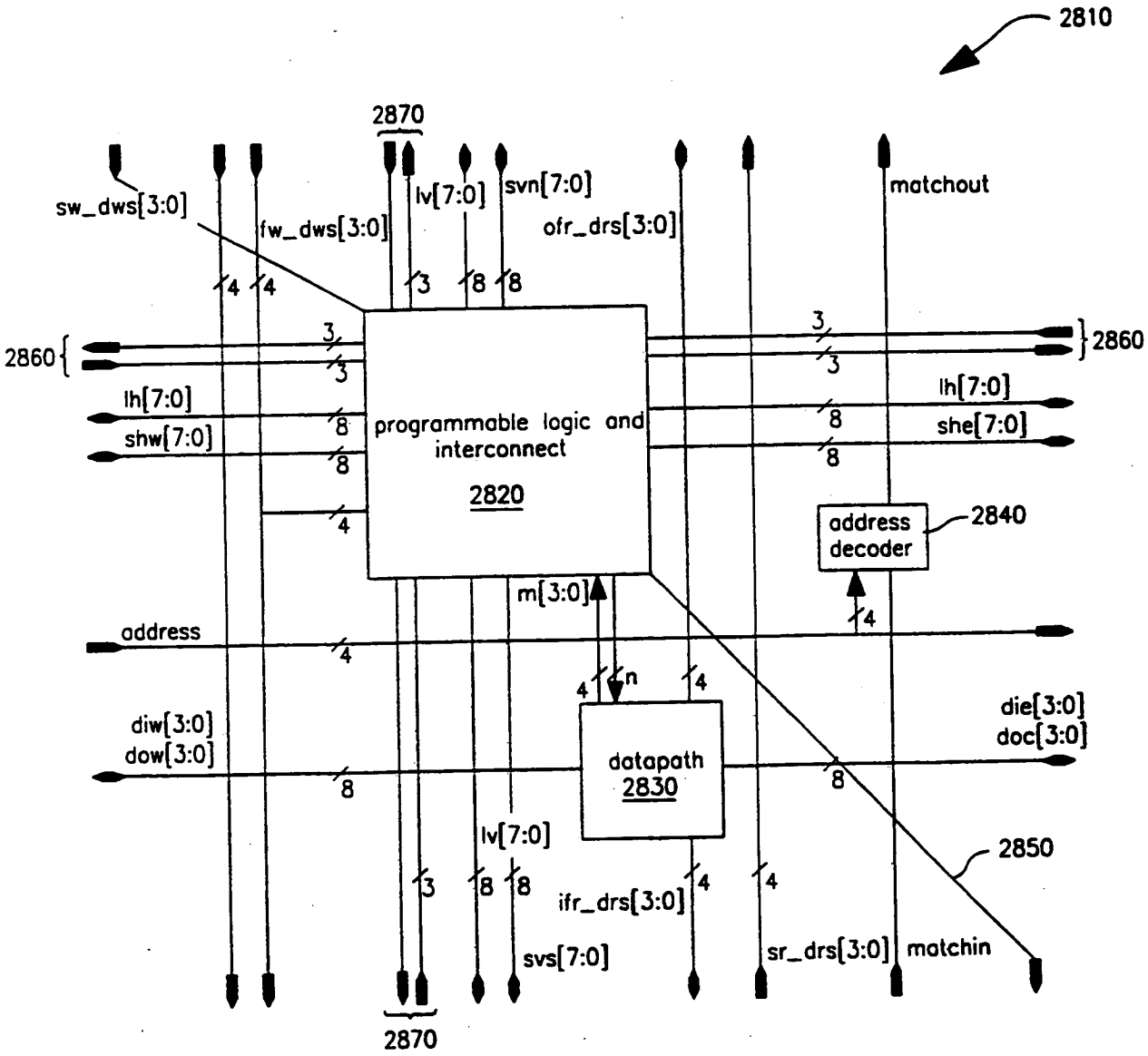

2810
2870
sw_dws[3:0]
fw_dws[3:0]
lv[7:0]
svn[7:0]
ofr_drs[3:0]
matchout
2860
2860
lh[7:0]
shw[7:0]
programmable logic and interconnect
2820
lh[7:0]
she[7:0]
address decoder
2840
m[3:0]
address
n
die[3:0]
doc[3:0]
diw[3:0]
dow[3:0]
datapath
2830
lv[7:0]
2850
ifr_drs[3:0]
svs[7:0]
sr_drs[3:0]
matchin
2870

FIG. 28

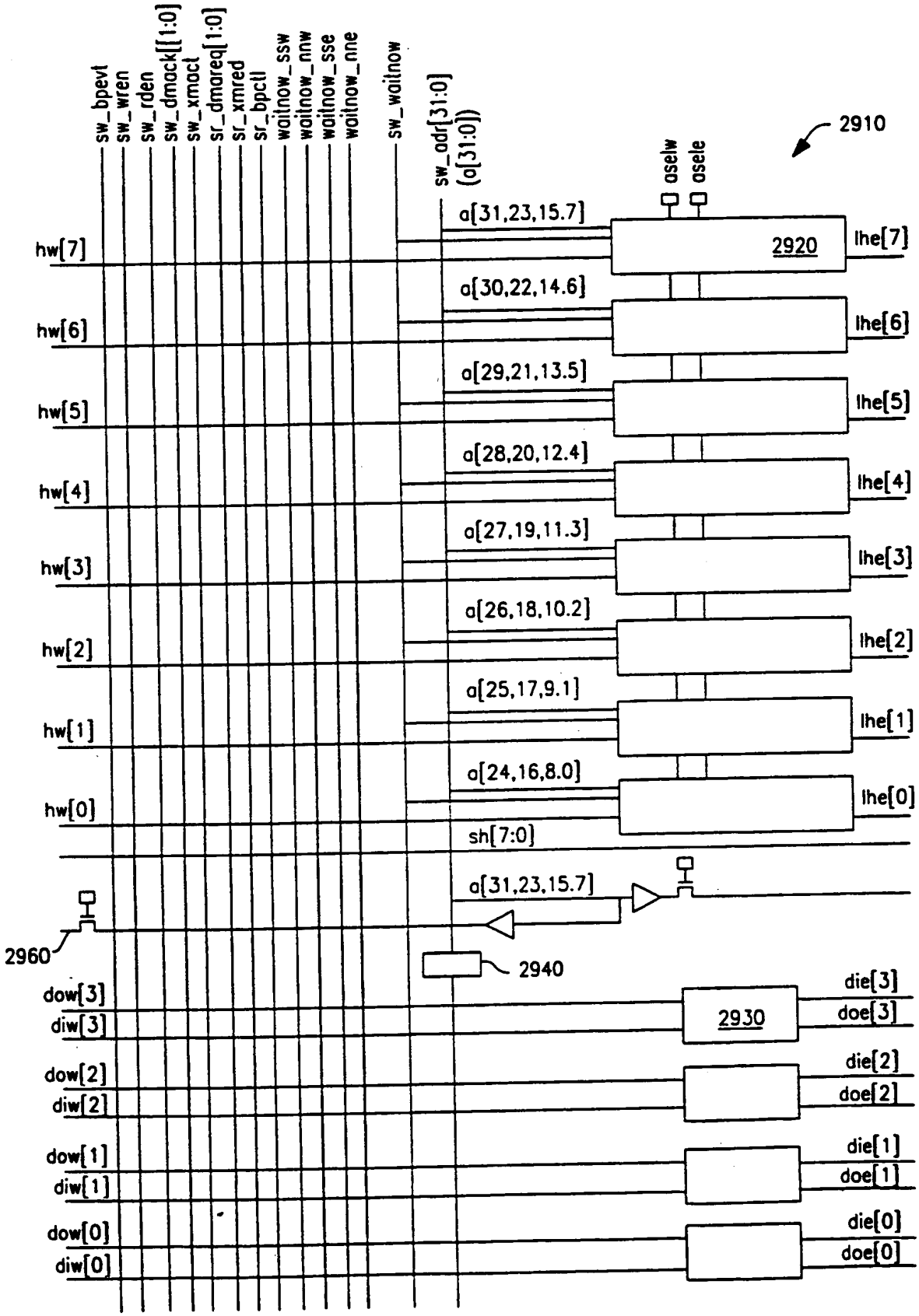

sw_bpevt
sw_wren
sw_rden
sw_dmack[1:0]
sw_xmact
sr_dmareq[1:0]
sr_xmred
sr_bpctl
waitnow_ssw
waitnow_nnw
waitnow_sse
waitnow_nne
sw_waitnow
sw_adr[31:0]
(a[31:0])
2910
aselw
asele
a[31,23,15.7]
a[30,22,14.6]
a[29,21,13.5]
a[28,20,12.4]
a[27,19,11.3]
a[26,18,10.2]
a[25,17,9.1]
a[24,16,8.0]
hw[7]
hw[6]
hw[5]
hw[4]
hw[3]
hw[2]
hw[1]
hw[0]
2920
lhe[7]
lhe[6]
lhe[5]
lhe[4]
lhe[3]
lhe[2]
lhe[1]
lhe[0]
sh[7:0]
a[31,23,15.7]
2960
2940
dow[3]
diw[3]
dow[2]
diw[2]
dow[1]
diw[1]
dow[0]
diw[0]
2930
die[3]
doe[3]
die[2]
doe[2]
die[1]
doe[1]
die[0]
doe[0]

FIG. 29A

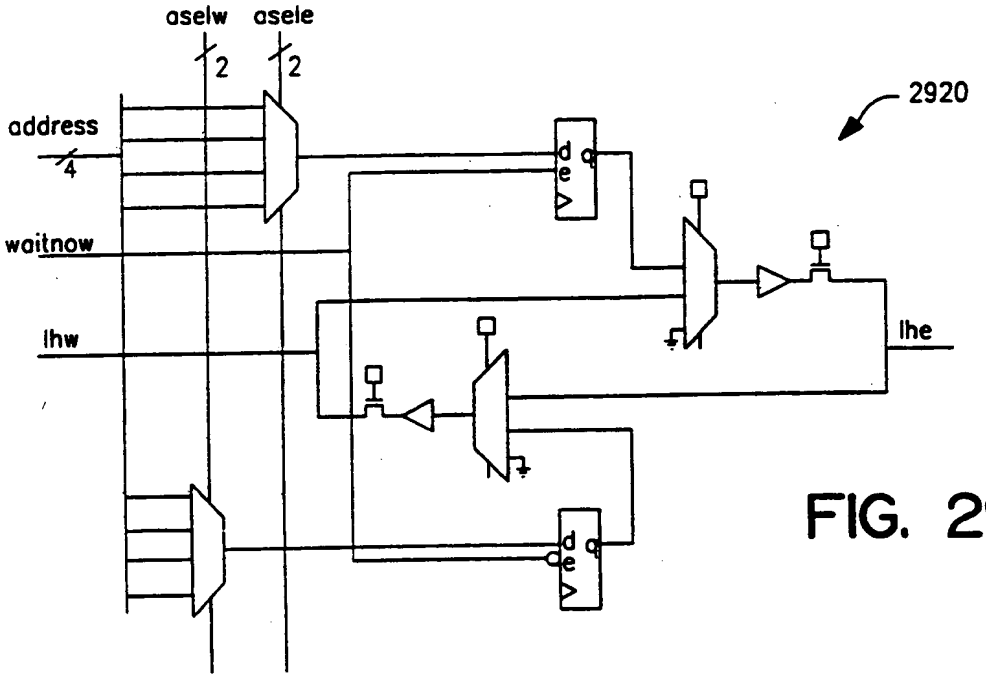


FIG. 29B

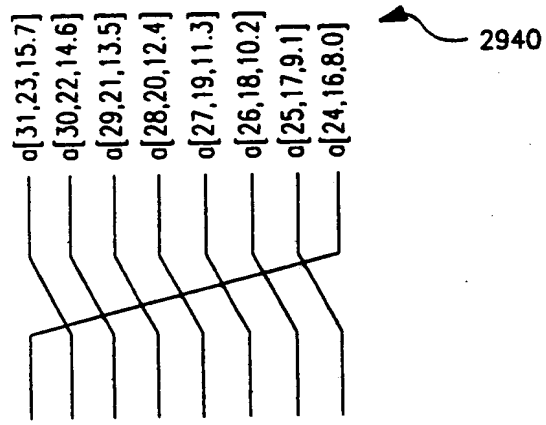


FIG. 29C

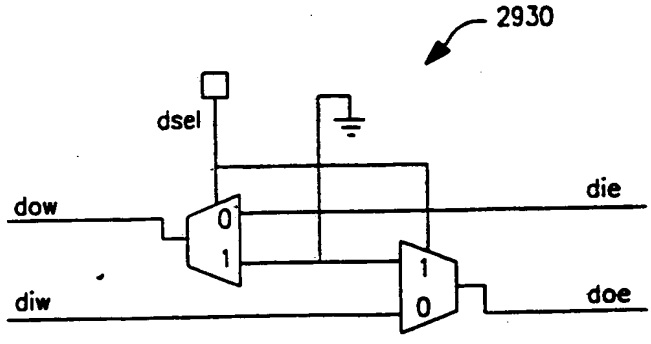


FIG. 29D

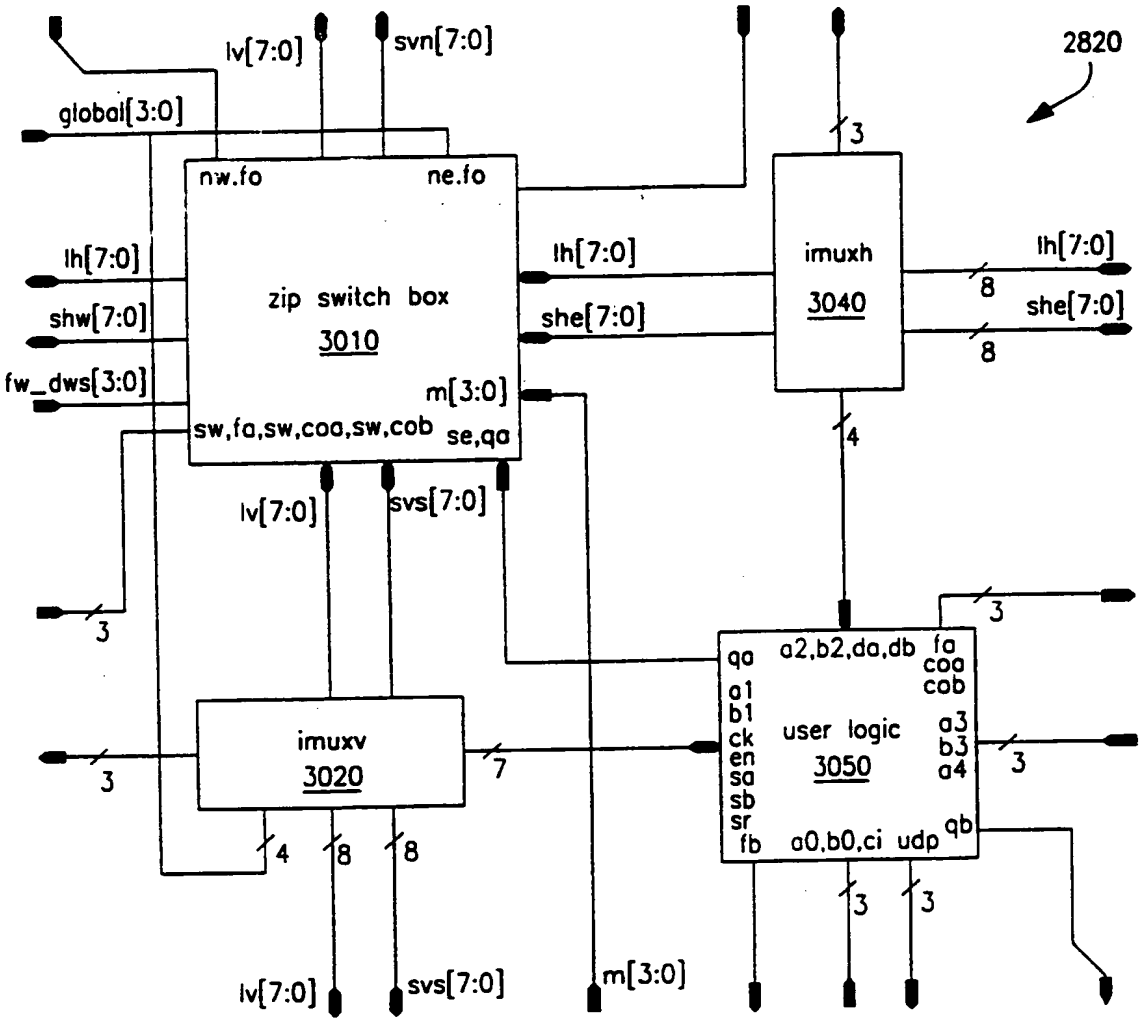


FIG. 30A

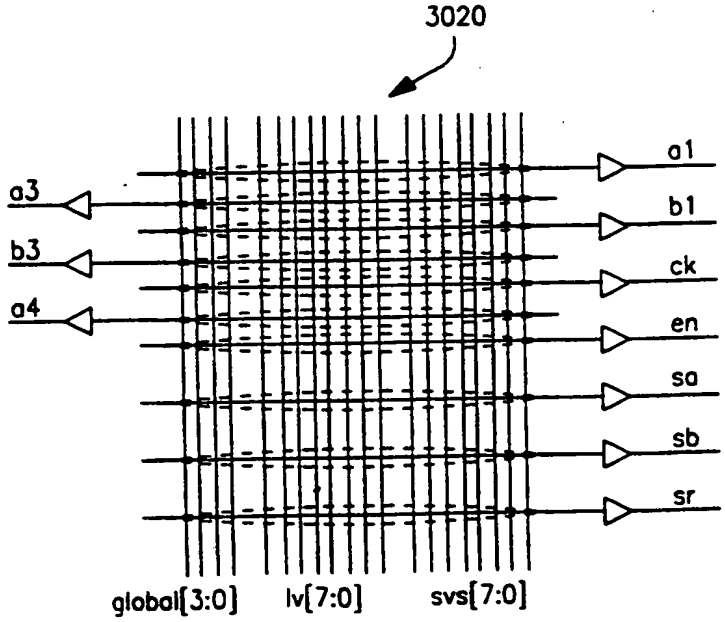


FIG. 30B

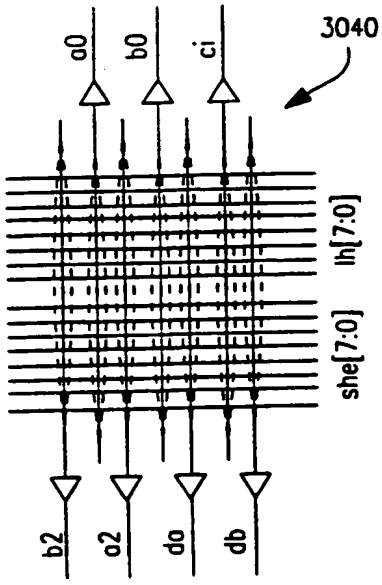


FIG. 30C

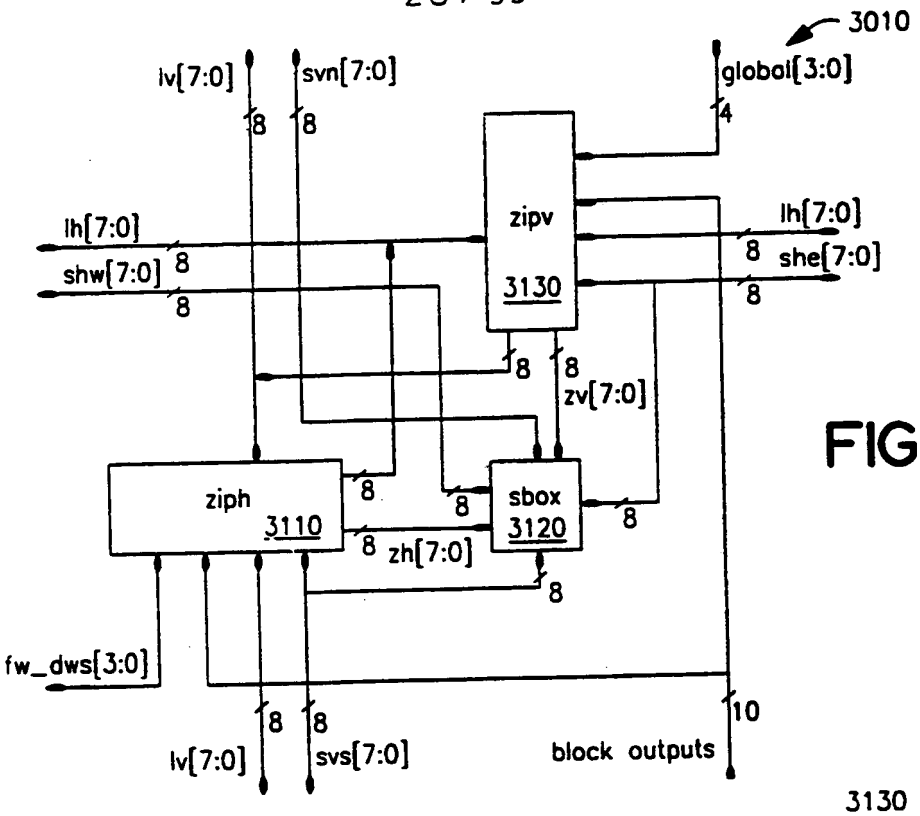


FIG. 31A

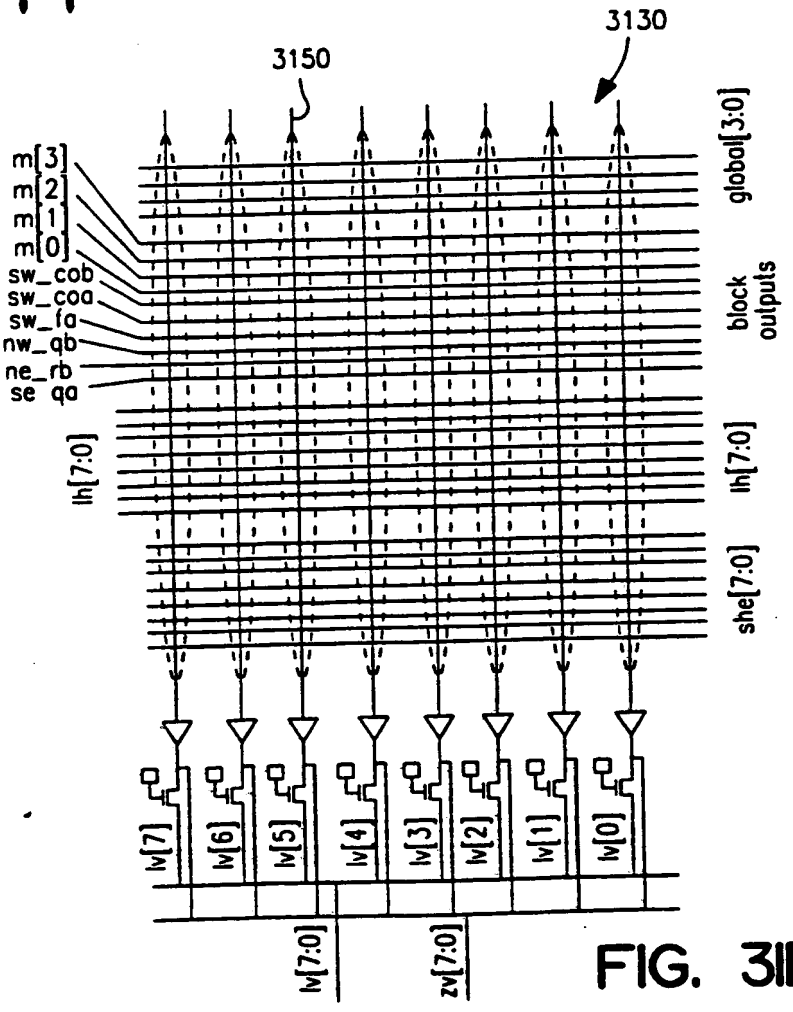


FIG. 31B

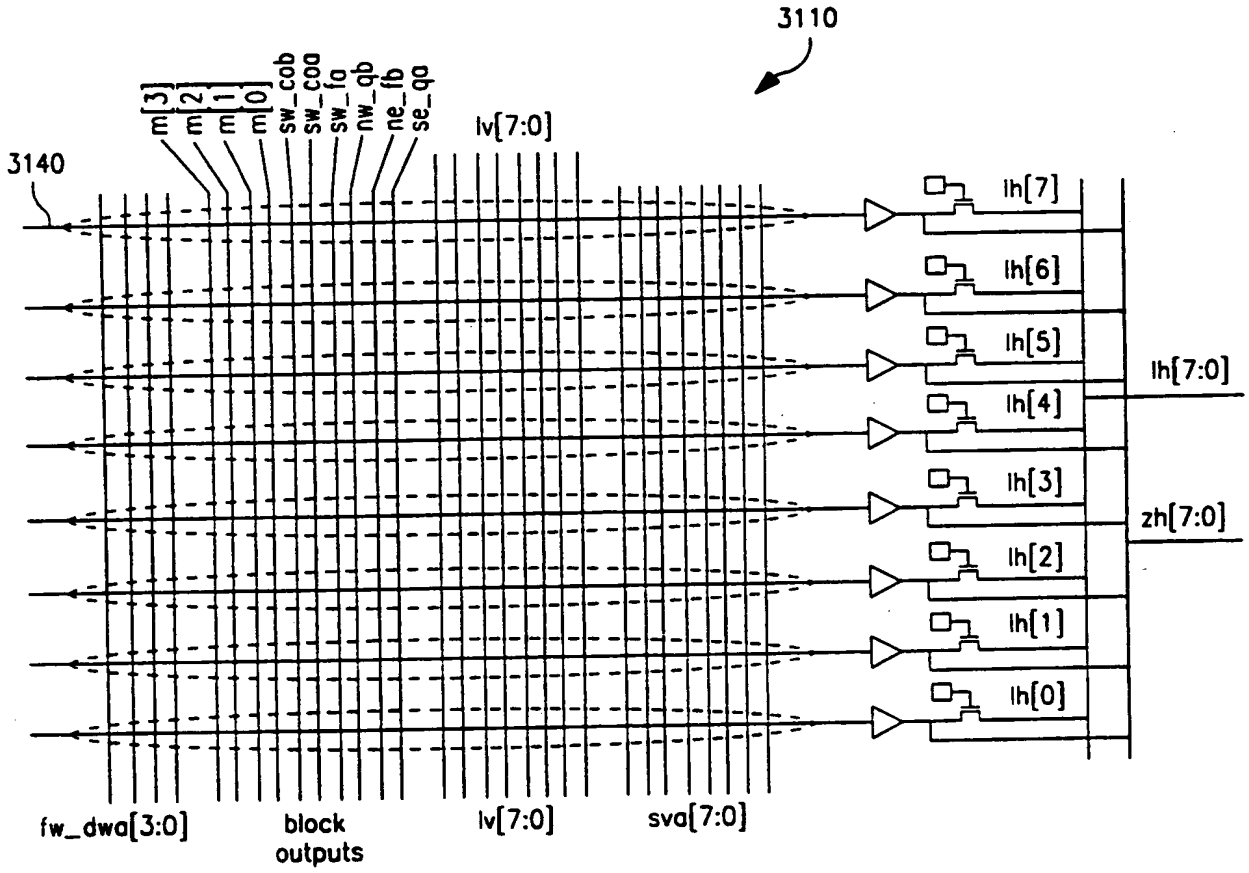

3110
3140
m[3]
m[2]
m[1]
m[0]
sw_cob
sw_coa
sw_fa
nw_qb
ne_fb
se_qa
lv[7:0]
lh[7]
lh[6]
lh[5]
lh[4]
lh[3]
lh[2]
lh[1]
lh[0]
lh[7:0]
zh[7:0]
fw_dwa[3:0]
block outputs
lv[7:0]
sva[7:0]

FIG. 31C

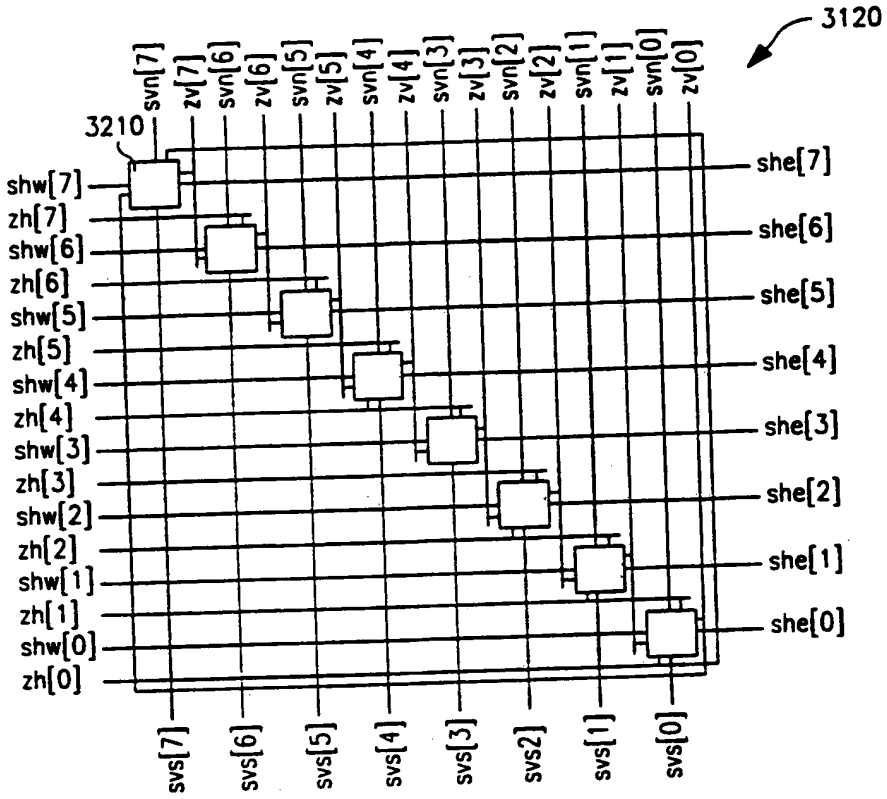


FIG. 32A

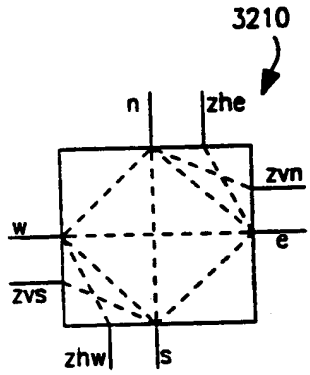
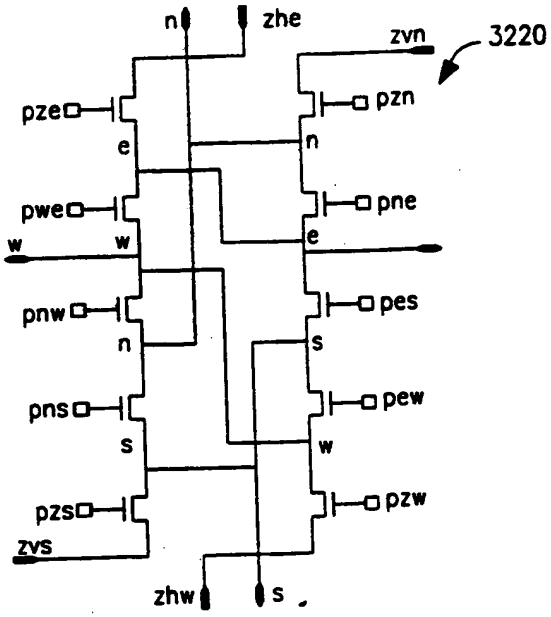
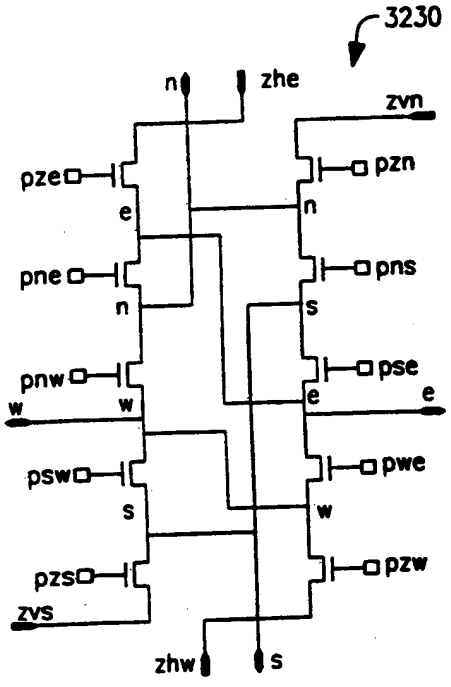


FIG. 32B



ten-way switch (first implementation)

FIG. 32C



ten-way switch (second implementation)

FIG. 32D

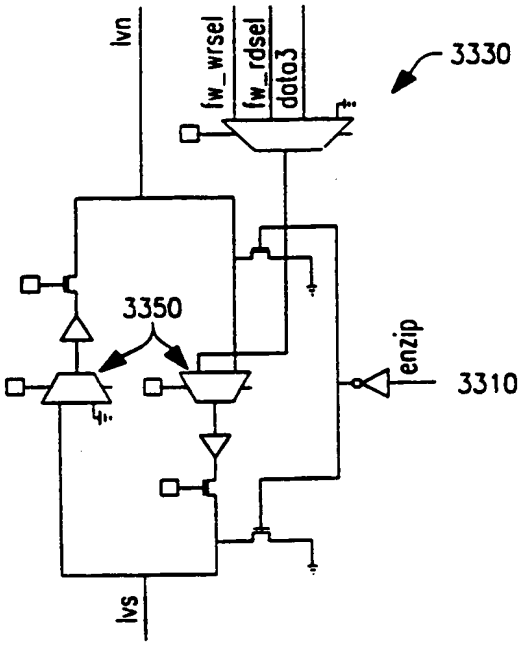


FIG. 33A

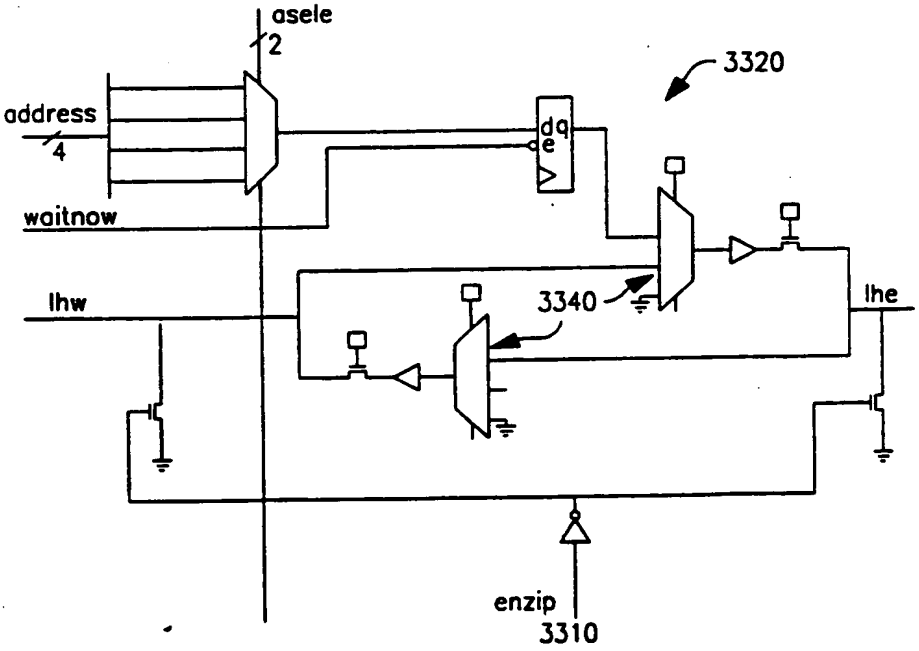


FIG. 33B

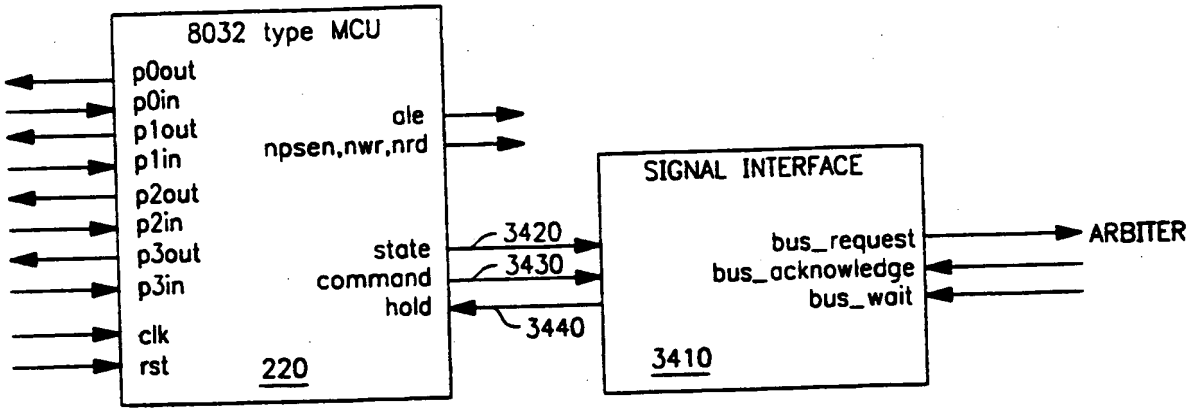

8032 type MCU
p0out
p0in
p1out
p1in
p2out
p2in
p3out
p3in
clk
rst
220
ale
npsen,nwr,nrd
state
command
hold
3420
3430
3440
SIGNAL INTERFACE
bus_request
bus_acknowledge
bus_wait
3410
ARBITER

FIG. 34

FIG. 35A

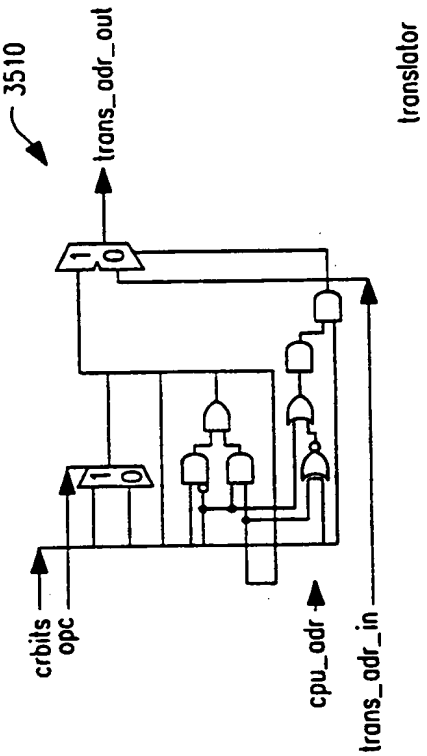
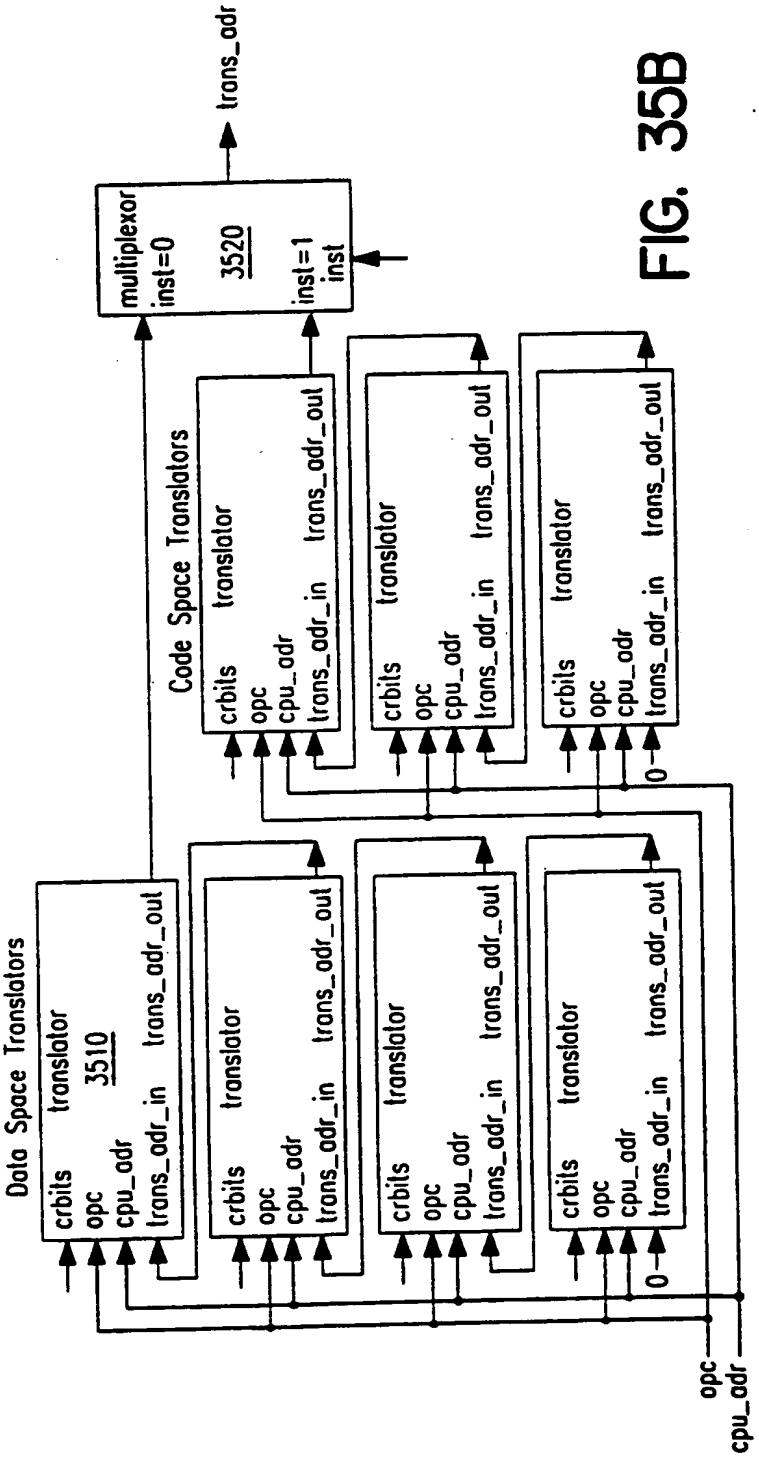


FIG. 35B



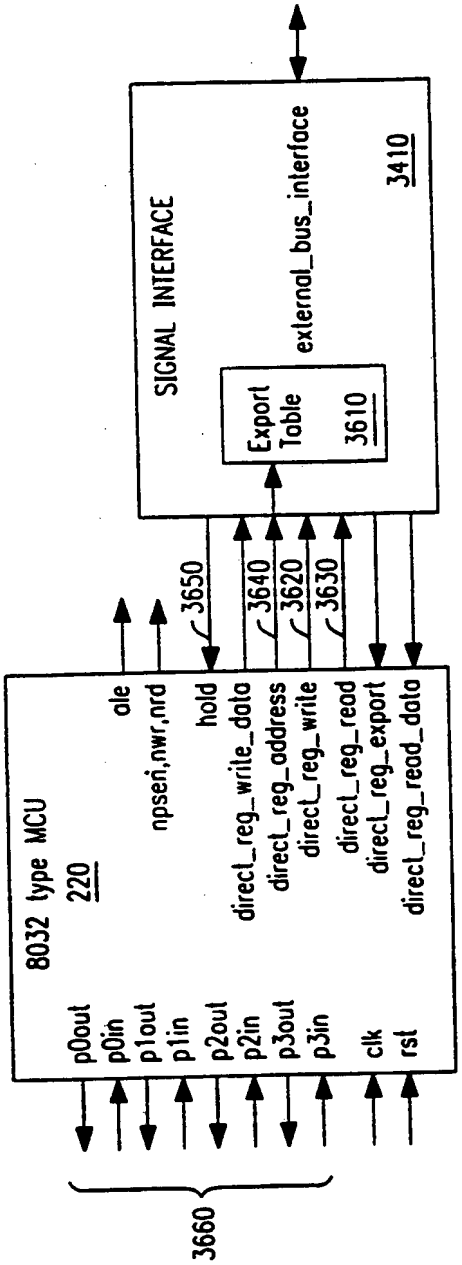

SIGNAL INTERFACE
external_bus_interface
3410
Export Table
3610
3650
3640
3620
3630
8032 type MCU
220
ale
npsen,nwr,nrd
hold
direct_reg_write_data
direct_reg_address
direct_reg_write
direct_reg_read
direct_reg_export
direct_reg_read_data
p0out
p0in
p1out
p1in
p2out
p2in
p3out
p3in
clk
rst
3660

FIG. 36

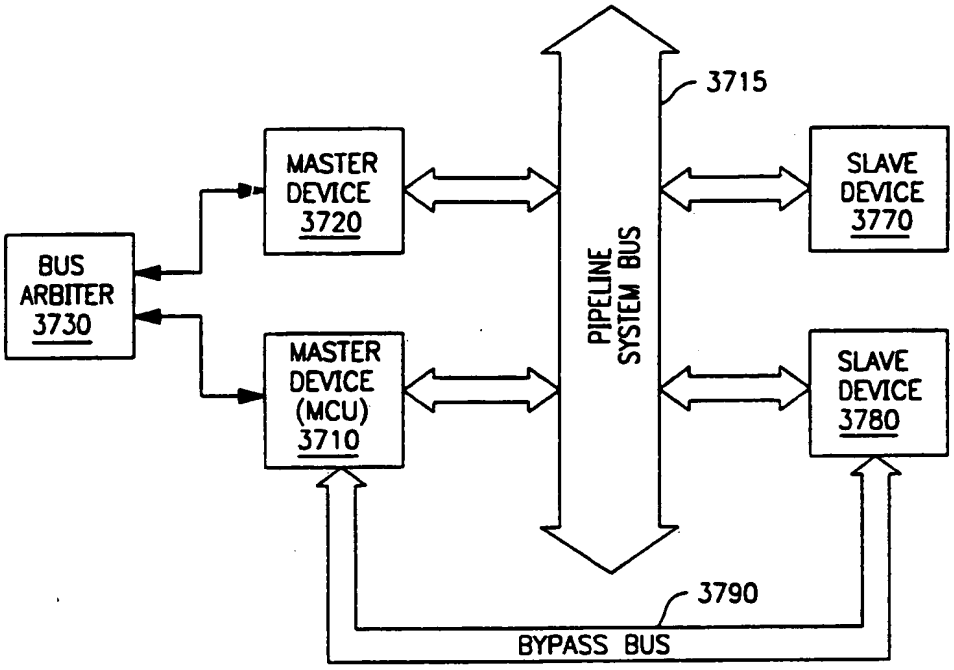

3715
MASTER DEVICE 3720
SLAVE DEVICE 3770
BUS ARBITER 3730
PIPELINE SYSTEM BUS
MASTER DEVICE (MCU) 3710
SLAVE DEVICE 3780
3790
BYPASS BUS

FIG. 37

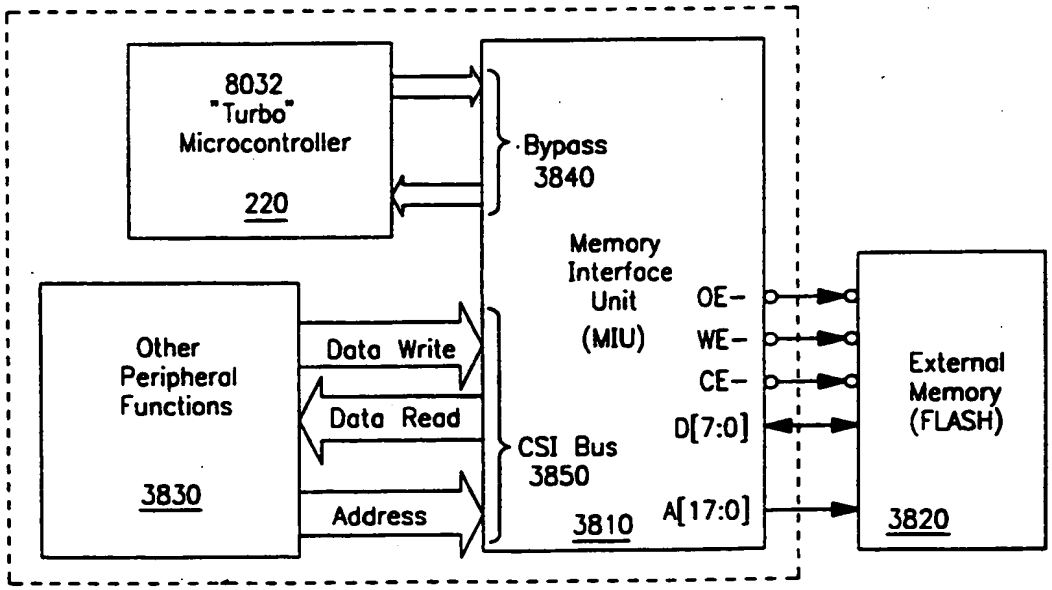

8032 "Turbo" Microcontroller 220
Bypass 3840
Memory Interface Unit (MIU)
OE-
WE-
CE-
D[7:0]
A[17:0]
External Memory (FLASH) 3820
Other Peripheral Functions 3830
Data Write
Data Read
Address
CSI Bus 3850
3810

FIG. 38

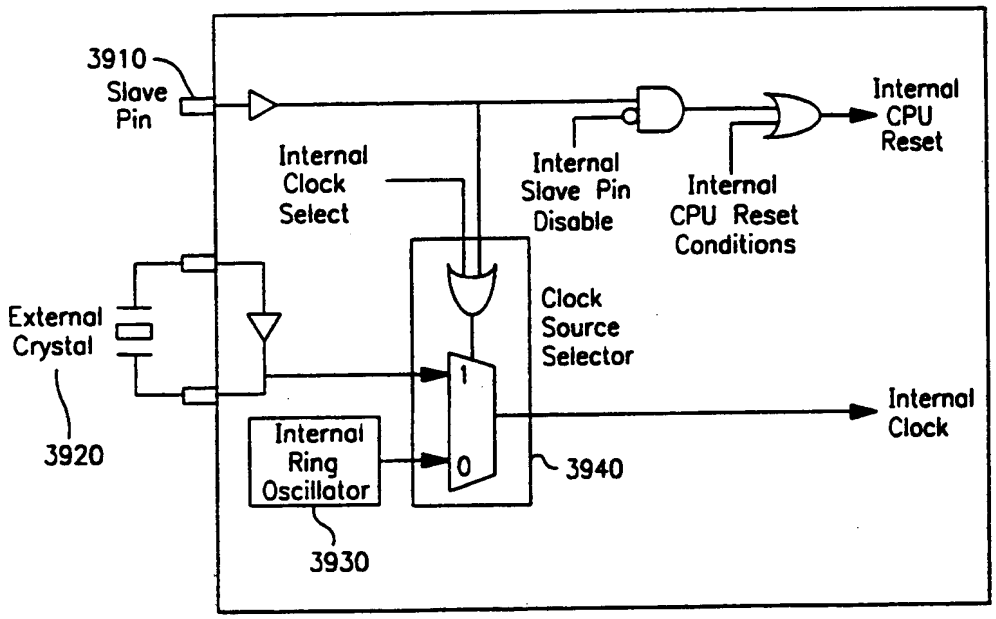

3910
Slave Pin
Internal Clock Select
Internal Slave Pin Disable
Internal CPU Reset Conditions
Internal CPU Reset
External Crystal
3920
Clock Source Selector
1
0
Internal Clock
Internal Ring Oscillator
3930
3940

FIG. 39

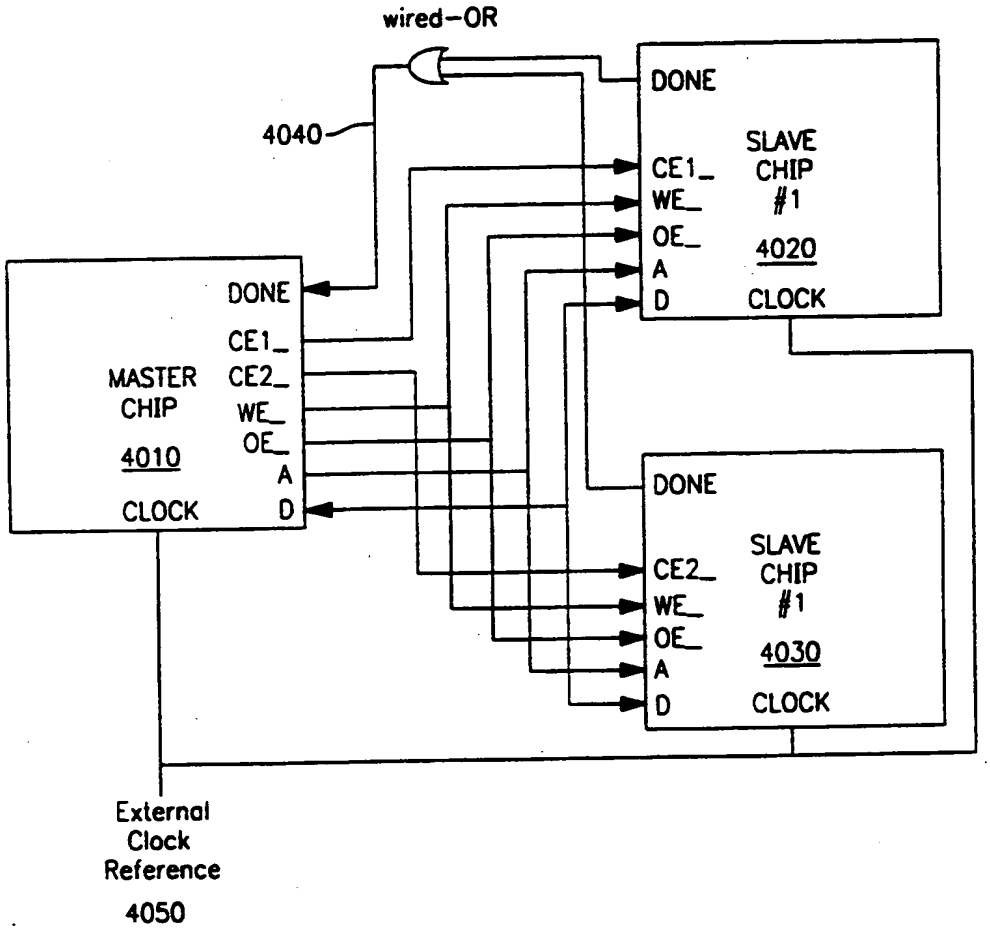

wired-OR
4040
DONE
CE1_
WE_
OE_
A
D
SLAVE CHIP #1
4020
CLOCK
DONE
CE2_
WE_
OE_
A
D
SLAVE CHIP #1
4030
CLOCK
MASTER CHIP
4010
DONE
CE1_
CE2_
WE_
OE_
A
D
CLOCK
External Clock Reference
4050

FIG. 40

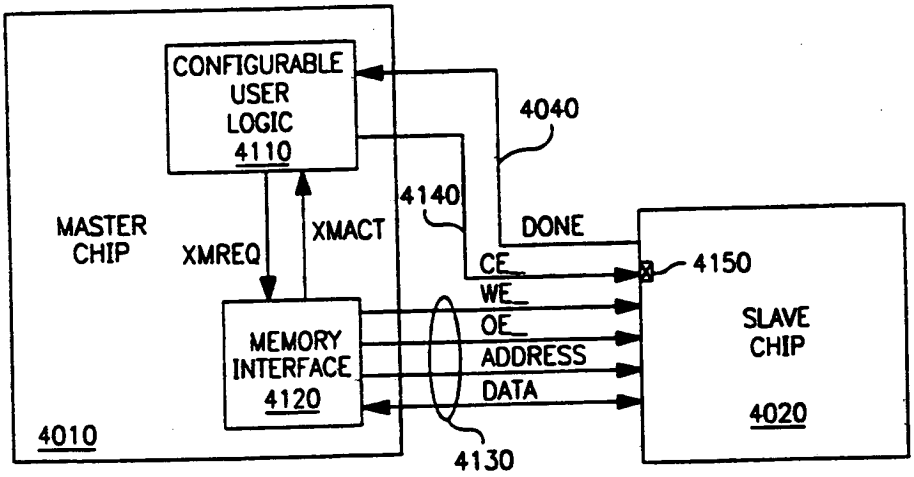

CONFIGURABLE USER LOGIC 4110
4040
4140
MASTER CHIP
XMREQ
XMACT
DONE
CE
WE
OE
ADDRESS
DATA
4150
MEMORY INTERFACE 4120
SLAVE CHIP
4020
4010
4130

FIG. 41

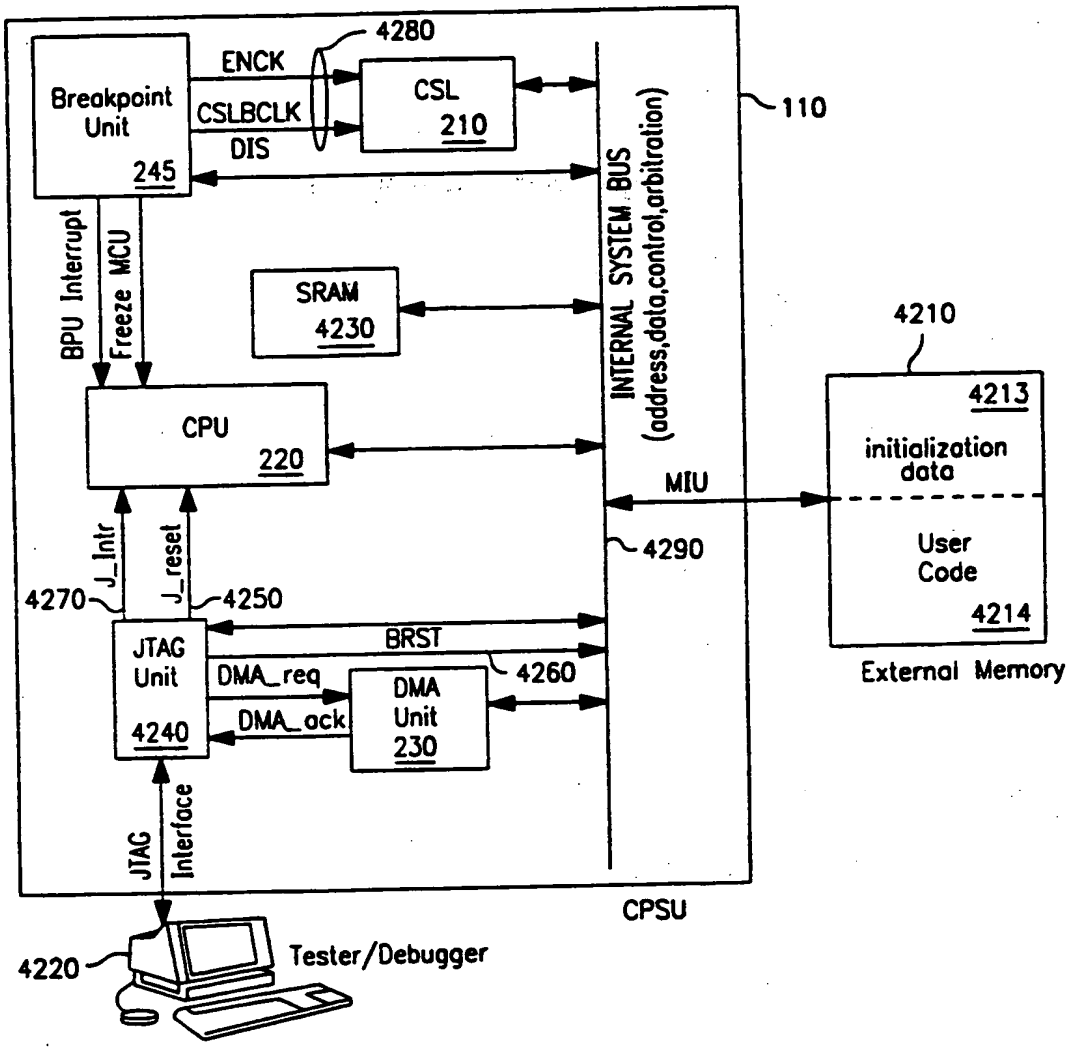

4280
ENCK
CSLBCLK
DIS
Breakpoint Unit
245
CSL
210
110
INTERNAL SYSTEM BUS
(address,data,control,arbitration)
BPU Interrupt
Freeze MCU
SRAM
4230
4210
4213
initialization data
CPU
220
MIU
User Code
4214
External Memory
4290
J_Intr
J_reset
4270
4250
BRST
JTAG Unit
4240
DMA_req
DMA_ack
4260
DMA Unit
230
JTAG Interface
CPSU
4220
Tester/Debugger

FIG. 42